UNIVERSIDADE TÉCNICA DE LISBOA

INSTITUTO SUPERIOR DE ECONOMIA E GESTÃO

MESTRADO EM: Decisão Económica e Empresarial

DESENVOLVIMENTO DE APLICAÇÃO NO EXCEL PARA O ESTUDO DE MÉTODOS HEURÍSTICOS

JORGE OLIVEIRA DA COSTA NEVES

Orientação:

Professora Doutora Maria Margarida de Oliveira Moz Carrapa

Professora Doutora Maria Cândida Vergueiro Monteiro Cidade Mourão

Júri:

Professora Doutora Margarida Maria Gonçalves Vaz Pato

Professora Doutora Maria Margarida de Oliveira Moz Carrapa

Professora Doutora Maria Cândida Vergueiro Monteiro Cidade Mourão

Professora Doutora Leonor Almeida Leite Santiago Pinto.

Fevereiro 2011

RESUMO

Neste trabalho pretende-se dar um contributo para a aplicabilidade dos métodos heurísticos. O objectivo é desenvolver uma ferramenta denominada MetaHeur que funciona como suplemento do Microsoft Excel (MSExcel) que vai permitir ao utilizador aplicar algumas metaheurísticas a alguns problemas de optimização combinatória tipo Saco-Mochila (knapsack) e Caixeiro-Viajante (TSP) e desta forma gerar, em tempo computacional aceitável, uma solução.

O problema é carregado de forma amigável no MSExcel através de uma tabela. O utilizador com o recurso a um painel específico dependente do tipo de problema selecciona o algoritmo proposto, os parâmetros específicos para a metaheurística definida e os critérios mais genéricos de paragem. Os resultados serão exibidos numa outra folha do mesmo livro do MSExcel onde foi carregada a tabela com os dados do problema.

As Meta-heurísticas são descritas e é incluído o pseudocódigo respectivo, bem como as principais decisões tomadas, permitindo desta forma, que a sua análise e eventuais alterações possam ser efectuadas mais facilmente. Também se descreve um algoritmo genético para o problema de Saco-Mochila que poderá ser facilmente adaptado para o problema do Caixeiro-Viajante. Pretende-se facilitar a quem esteja interessado, a sua posterior codificação de modo a melhorar o Metaheur.

ABSTRACT

This work intends to make a contribution to the applicability of heuristic methods. The aim is to develop a tool called MetaHeur that works as a supplement of Microsoft Excel (MSExcel). This will allow the user to apply some meta-heuristics, in a friendly way, for some combinatorial optimization problems, such as Knapsack and Travelling Salesman and thus obtaining "good" solutions, in a reasonable computational time.

The problem is friendly loaded in MSExcel, through a table, and the user selects a specific panel depending of the problem and a proposed algorithm. Based on the algorithm selects the specific parameters, more general parameters of stop criteria and results will be displayed on another sheet of the same MSExcel book where the table with the data of the problem was loaded.

The metaheuristics are described and the respective pseudocode is included, thus enabling its analysis and the possibility of introducing more easily any changes. A genetic algorithm for the Knapsack problem is also described, which can easily be adapted to the Traveling Salesman problem in order to facilitate those that are interested to improve MetaHeur in the future with subsequent codification.

AGRADECIMENTOS

Em primeiro lugar gostaria de expressar os meus sinceros agradecimentos às Professoras Doutoras Margarida Moz e Cândida Mourão, pela orientação cuidadosa, paciência, amizade e pelas valiosas sugestões.

Agradecimentos às Professoras Doutoras Margarida Pato e Leonor Santiago Pinto pelas valiosas sugestões e apoio.

Agradecimento a todos os que me deram apoio para a realização deste trabalho e em especial:

- a todos os Professores do mestrado de DEE (Decisão Económica e Empresarial) do ISEG pelo apoio e conhecimentos transmitidos;
- ao meu filho Bernardo pela ajuda na elaboração do algoritmo genético;
- ao Dr. Abel Vinagre pelo seu apoio;
- à minha mulher Ângela pelo amor e estímulo;
- aos meus pais Margarida e Telmo por me facilitarem a vida;
- à minha maninha Ana e restante família pelo auxílio dado;
- a todos os colegas e amigos que directamente ou indirectamente compartilharam comigo este percurso académico nomeadamente o Henrique Dias.

ÍNDICES

Figuras

Tabelas

Anexo: CD contendo o MetaHeur e ficheiros com exemplos

INTRODUÇÃO

Este projecto está inserido no plano curricular do Mestrado de Decisão Económica e Empresarial, no 2º ano / 2º semestre.

A ideia surge baseada no suplemento *Solver* cujo sucesso e utilidade tem sido amplamente divulgado, essencialmente por ser um recurso do MSExcel e por tirar proveito da facilidade de inserir dados, trabalhá-los e desenvolver relatórios e gráficos de uma forma extremamente simples.

Após as aulas da disciplina de Heurísticas leccionada no âmbito do mestrado referenciado, ficou praticamente assente a ideia de poder haver algum interesse na criação de uma ferramenta para funcionar eventualmente como apoio pedagógico para futuros alunos de heurísticas.

Assim surge o nome da aplicação **MetaHeur,** correspondente às oito letras iniciais de Metaheurísticas.

Foi desenvolvida neste trabalho a aplicação MetaHeur para resolver, recorrendo a metaheurísticas, os problemas do Saco-Mochila Binário (Binary Knapsack Problem) (Glover,1989;Hillier e Liberman, 2010) e do Caixeiro-Viajante (TSP - Travelling Salesman Problema) (Glover,1989;Hillier e Liberman, 2010).

O desenvolvimento do MetaHeur não deve ser encarado como um fim em si mesmo, mas sim a possibilidade de se iniciar um projecto que envolva vários alunos que complementem o produto com mais tipos de problemas e mais algoritmos, por isso, deve ser visto como um produto inicial que possa e deva ser complementado posteriormente. Por essa razão o código encontra-se aberto e preparado para receber mais algoritmos e tipos de problemas adicionais desenvolvidos por terceiros.

Espera-se que este facto promova o interesse no seu desenvolvimento e não se limite a ser visto como um trabalho efectuado no âmbito de um mestrado.

1. METAHEUR COMO SUPLEMENTO DO MICROSOFT EXCEL E SUAS CARACTERÍSTICAS GENÉRICAS

Um dos maiores problemas que os utilizadores podem encontrar é o de ser fastidiosa a introdução dos dados referentes a diversos tipos de problemas, como por exemplo o Caixeiro-Viajante, principalmente se o número de cidades for elevado. Torna-se assim primordial, efectuar um desenvolvimento que permita aproveitar as tabelas de dados que estejam em ficheiros, em bases de dados ou num outro formato importável pelo Excel. Da mesma forma, parece ser importante que os *outputs* gerados pelo MetaHeur possam ser posteriormente exportados para relatórios onde o utilizador possa facilmente manipular os resultados e os gráficos.

Nesta conformidade, o desenvolvimento das interfaces do programa para a importação de dados e posteriormente para exportação dos resultados, obrigaria a um esforço de programação e de horas de trabalho elevado que iria consumir as mesmas horas ou ainda mais do que a programação dos algoritmos, razão deste trabalho.

Sabe-se que o Excel é uma ferramenta de produtividade pessoal dominada por grande parte dos utilizadores das metaheurísticas e dos utilizadores em geral. É muito intuitivo, a inserção de dados é fácil e tem excelente capacidade de importação de dados (já desenvolvida pela Microsoft). Por outro lado, a integração de dados com os outros programas que integram o Microsoft Office, tais como o Word, está extremamente desenvolvida e por conseguinte, parece ser o produto ideal para servir de suporte ao MetaHeur. Tanto assim é, que o *Solver* se utiliza e bem, como suplemento do Excel, tornando-se num sucesso nesta área. Este ponto parece ser relevante, pois os utilizadores do *Solver* fazem parte do público-alvo a que se destina o MetaHeur, tornando-se importante aproveitar o *know how* entretanto adquirido.

Considerando ainda que não está em jogo o desempenho do MetaHeur, isto é, a sua capacidade de processamento da informação, mas sim os aspectos anteriormente enunciados, parece que o produto ideal para o funcionamento integrado é o Excel.

Como vantagem, e à semelhança do *Solver*, pode desenvolver-se o MetaHeur como um suplemento do Excel, facilitando a sua instalação e operacionalidade.

Resumindo, a adopção do MetaHeur como suplemento do MSExcel resulta nas seguintes vantagens:

- facilidade de instalação;
- facilidade na importação de dados;

- facilidade na construção de tabelas com os dados do problema;
- suporta uma linguagem de programação VBA (*Visual Basic for Applications)* (Green at al 2007);
- suporta macros, ou seja, a gravação de tarefas que são executadas repetidamente;
- interage com todas as aplicações do MS Office, tais como Access, Word, etc…;
- funciona de forma semelhante ao *Solver*;
- permite, com recurso aos programas do MS Office, melhorar os *outputs* gerados pelo MetaHeur de forma fácil e intuitiva;
- permite guardar toda a informação utilizando as funções do MSExcel disponíveis para esse efeito, incluindo vários tipos de formatos de ficheiro, com segurança através de *password,* etc….

A evidência desta boa solução encontrada para o desenvolvimento do MetaHeur não invalida que se pretenda melhorar o nível de interacção com os utilizadores do Excel, por exemplo:

- o MetaHeur deve-se inserir de uma forma intuitiva e respeitando o *design* do MSExcel;
- a leitura dos dados por parte do MetaHeur deverá ser feita o mais simples possível, não sendo necessário definir toda a tabela, mas apenas as células iniciais de cada tabela;
- ser capaz de identificar tabelas de dados simétricas que se possam gerar automaticamente;
- o tipo de problema, o algoritmo e os parâmetros quer gerais, quer específicos do algoritmo seleccionado deverão ser intuitiva e rapidamente percepcionados;
- de acordo com as selecções efectuadas, se for necessário, devem exibir as opções disponíveis. Só estas deverão ser listadas e não uma lista fastidiosa para facilitar a interacção com o utilizador;
- é importante que exista uma opção que permita gerar automaticamente uma solução inicial admissível, ou optar pela introdução de uma tabela que contenha a solução inicial que se entenda que deva ser o ponto de partida para a aplicação das metaheurísticas;
- o MetaHeur deve proporcionar rotinas de correcção automática dos dados introduzidos e dialogar interactivamente com o utilizador para que valide a referida correcção automática proposta pelo MetaHeur. Por exemplo, no caso de um problema do tipo Saco-Mochila binário, se a solução inicial tiver um valor não binário, o programa deverá avisar que esse valor vai ser corrigido para o valor 0, pedindo para o efeito a confirmação por parte do utilizador.

Em suma, estas foram as opções tomadas ao escolher o MetaHeur para funcionar como suplemento do Excel.

O MetaHeur, como se detalhará nos capítulos seguintes, tem duas fases fundamentais:

1. Identificação de uma solução admissível inicial, que pode ser introduzida pelo utilizador ou gerada de forma aleatória;
2. Utilização de uma metaheurística para identificação de uma solução admissível de boa qualidade. Foi desenvolvida uma GRASP (Feo e Resende, 1995) e uma Tabu-Search (Glover, 1989) para cada um dos problemas. Também se desenvolveu, em pseudocódigo, um Algoritmo Genético (Russel e Norvig, 2004) que poderá ser mais tarde codificado.

2. SOLUÇÃO ADMISSÍVEL INICIAL

2.1. Introdução

Em qualquer dos dois problemas (Saco-Mochila e Caixeiro-Viajante) deve-se poder optar entre introduzir uma solução inicial, ou permitir que seja o próprio MetaHeur a fazer a sua geração aleatória, como se ilustra na figura 1.

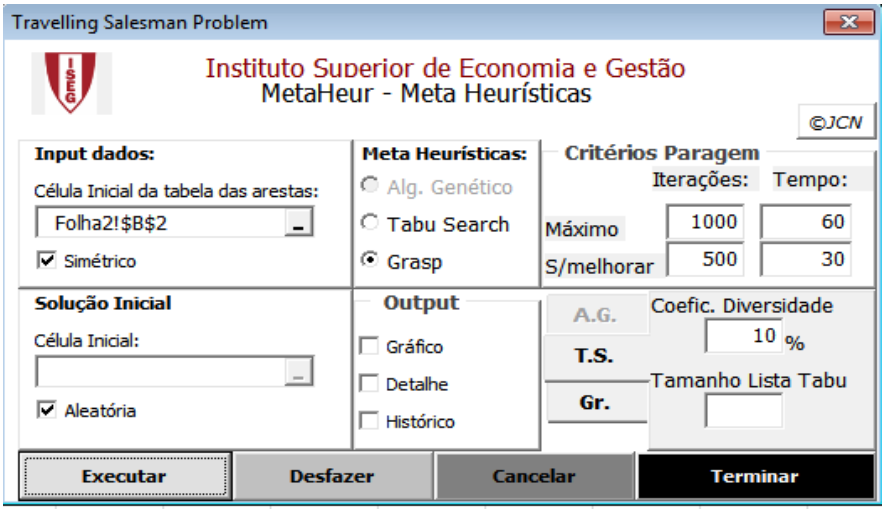


FIGURA 1 - PAINEL TSP

Como ponto prévio devemos assumir que o Metaheur carrega os dados em 3 vectores, conforme o tipo de problema, de acordo com a Tabela 1.

Tipo Problema	Coluna 1 Tabela	Coluna 2 Tabela	Coluna 3 Tabela
Saco-Mochila	Elemento identificativo	Agradabilidade	Peso
Caixeiro-Viajante	Vértice inicial	Vértice final	Distância

TABELA 1 - ESTRUTURA DE CARREGAMENTO DA TABELA DE DADOS

No caso do problema do Caixeiro-Viajante, uma diferença fundamental no carregamento dos vectores está relacionada com a orientação dos arcos. Caso se pretenda uma rede simétrica, o MetaHeur duplica automaticamente os arcos não representados, segundo o procedimento "Duplica Arestas".

Após o carregamento de um arco na linha i , coluna 1, 2 e 3 da tabela de dados introduzidos é criada um novo arco imediatamente a seguir onde o vértice inicial será o vértice final do arco anterior, o vértice final será o inicial do arco anterior e o peso igual ao do arco anterior.

Procedimento Duplica Arestas

Entradas: CélulaLinha i, Opção simétrica

Se Simétrica Activada **Então**

i ← i + 1

coluna 1 tabela (i) = coluna 2 tabela (i-1)

coluna 2 tabela (i) = coluna 1 tabela (i-1)

coluna 3 tabela (i) = coluna 3 tabela (i-1)

Fim

Note-se que não há qualquer preocupação quanto à eventual introdução de arcos repetidos que poderão provocar uma tabela com mais arcos do que seria necessário, uma vez que não serão consideradas. O programa utiliza os arcos por ordem e se este já estiver a ser usado não é considerado, mesmo que a distância seja menor, uma vez que quando um arco é seleccionado, deixam de estar disponíveis para serem seleccionados todos os arcos que tenham os mesmos vértices de partida e os mesmos de chegada. No entanto, estando bastante facilitada a sua introdução, será conveniente que se efectue uma conferência aos dados introduzidos, usando para o efeito, as ferramentas disponibilizadas pelo Excel.

Explica-se nos pontos seguintes, os procedimentos de leitura de dados e de geração aleatória da solução inicial.

2.2. Leitura de Dados

Considerando o esquema de introdução de dados definido na Tabela 1, do ponto anterior, e de acordo com o tipo de problema, a tabela de dados deverá ser importada para qualquer folha do Excel e para qualquer célula dessa folha, seleccionadas pelo utilizador, no formato de 3 colunas definido.

O MetaHeur permite, à semelhança do Solver, que o utilizador escolha o local mais adequado para a colocação da sua tabela com os dados do problema. Por outro lado, também não se torna necessário definir a *range* da tabela, de modo a evitar ter de percorrer as tabelas extensas para que se identifique a célula de início e do fim da tabela.

Assim, basta apenas identificar a célula do canto superior esquerdo da tabela de dados (correspondente ao vértice inicial do primeiro arco no caso do Caixeiro-Viajante ou da célula identificadora do primeiro elemento no caso do Saco-Mochila). De seguida, o programa percorrerá essa tabela de três colunas, até encontrar uma célula vazia na primeira coluna. Este carregamento em memória é efectuado sempre com recurso aos três vectores definidos: “coluna 1 tabela”, “coluna 2 tabela”, “coluna 3 tabela”.

Considerando que a tabela de dados iniciais foi carregada nos três vectores correspondentes às colunas 1, 2 e 3, com os significados adequados a cada tipo de problema, escreve-se de seguida e de forma sucinta o procedimento de leitura de uma solução inicial e o respectivo pseudocódigo, no caso dos dois problemas em análise.

2.2.1. Leitura da Solução Inicial para o Problema do Saco-Mochila

Neste caso, o procedimento efectua a leitura do vector que contém os dados referentes a uma solução inicial do problema do Saco-Mochila, tendo em consideração os elementos lidos de acordo com a tabela 1. Para o efeito, considera-se que os índices do vector de identificação dos elementos serão os mesmos dos do vector solução, que é binário, isto é, o elemento i se é seleccionado tem o valor 1, caso contrário o valor 0.

Para a leitura da solução inicial, à semelhança do que é efectuado com os vectores de dados, o utilizador tem de informar qual a célula do Excel que determina a coluna com os valores binários

correspondentes à solução que pretende ler. De seguida, dá-se a indicação da célula do Excel que contém o valor da capacidade, ou seja, o termo independente da restrição do problema.

Com base nas entradas descritas, o procedimento executa, na mesma coluna, a leitura de todas as linhas e caso encontre um valor não binário, envia uma mensagem para confirmar a correcção automática desse valor. Por defeito esta correcção atribui o valor 0, ou seja, não selecciona o elemento da linha i, que corresponde ao elemento i do vector que identifica os elementos. Caso contrário, o utilizador pode alterar esta correcção forçando o valor 1.

Este processo só termina quando se encontra uma célula vazia.

Caso os tamanhos do vector dos elementos identificativos e solução inicial sejam diferentes, significa que houve um erro e que é necessário repetir o procedimento.

Para verificar se a solução é válida, é calculado o seu peso (após a validação anterior referente ao tamanho dos vectores identificação e solução) e verificado se é inferior à capacidade. Para o seu cálculo efectua-se a soma dos produtos do valor binário de cada elemento da solução pelo peso correspondente. Em simultâneo, calcula-se também e de acordo com o mesmo raciocínio a agradabilidade da solução.

Para a leitura do pseudocódigo deste procedimento, que se apresenta de seguida, considere-se que "célula" é uma célula do Excel, definida por linha e coluna. Por outro lado, a variável "linha" corresponde ao valor da linha referente à primeira célula da coluna de valores binários que representam a solução inicial.

Procedimento Introdução Solução Inicial Problema Knapsack

Entradas: Célula superior esquerda da tabela que contém os valores binários da solução inicial a introduzir, Capacidade.

Agradabilidade, Peso, soluçãoinicial() , i ← 0

Repita Até encontrar célula(linha+i,coluna) vazia

soluçãoinicial(i) ← célula(linha+i,coluna)

Se soluçãoinicial(i) <> 0 **e** soluçãoinicial(i) <>1 **Então**

x ← Mensagem(Valor inválido. Deseja corrigir para 0?)

Se x = Sim **Então**

soluçãoinicial(i) ← 0

Caso Contrário

Soluçãoinicial(i) ← 1

i ← i + 1

Se Tamanho da coluna 1 Tabela <> Tamanho vector soluçãoinicial **Então**

Mensagem(Tamanho incorrecto do vector dos elementos identificadores comparado com o tamanho do vector solução inicial)

Caso Contrário

Para i ← 0 **Até** tamanho vector soluçãoinicial **Faça**

Agradabilidade ← Agradabilidade + coluna 2 tabela (i) * soluçãoinicial(i)

Peso ← Peso + coluna 3 tabela (i) * soluçãoinicial(i)

Se Peso > Capacidade **Então**

Mensagem(Solução inicial proposta ultrapassa a capacidade, sugere-se a sua correcção ou a geração aleatória da solução inicial)

Próximo i

Próxima célula

Fim

2.2.2. Leitura da Solução Inicial para o Problema do Caixeiro-Viajante

Tal como no procedimento anterior relativo ao problema do Saco-Mochila, a solução inicial é um vector binário, lido em coluna, onde é indicada a célula com o primeiro valor do vector a ler. O procedimento percorre então as células das linhas abaixo até encontrar uma célula vazia, que se entende como sendo o fim do vector. Os seus valores, 0 ou 1, da linha i correspondem ao i-ésimo arco do mesmo índice, indicando o valor 1, que esta faz parte da solução.

Tal como no problema do Saco-Mochila, também neste caso se implementou o mecanismo de validação do vector binário, bem como no que se refere ao tamanho da solução lida.

Caso a validação não encontre erros, percorre-se o vector distância ("coluna 3 tabela" da Tabela 1) e multiplica-se pelo valor na célula correspondente da solução inicial. A soma destes cálculos dá a distância total da solução inicial.

De seguida há que verificar se a solução inicial é admissível. Para o efeito é executado um conjunto de procedimentos e funções que se descrevem de forma sucinta a seguir:

- Procedimento "número vértices"- tem como argumentos, os dois vectores lidos correspondentes às leituras das colunas 1 e 2 da matriz de dados, contendo respectivamente o vértice inicial e o vértice final de cada arco. A sua execução, percorrendo exaustivamente os vectores dos vértices inicias e finais, vai guardar no vector "vértices" todos os vértices diferentes encontrados;
- Vector "grauvértices"- vector com dimensão igual à do vector "vértices", com valores iguais a 1, 2 ou 3. A solução lida é percorrida e a um vértice é atribuído o valor 1, caso seja o vértice inicial do arco. O valor 2 identifica um vértice como extremidade final do arco. O valor 3 identifica vértices que tenham um arco a chegar e outro a partir. Desta forma será mais fácil controlar se um circuito é Hamiltoniano (Epp, 2004);
- Função "CirHam" tem como argumento o vector "grauvértices" e verifica se todos os vértices são do grau 3, isto é, se de todos sai um arco e entra outro. Caso seja verdade, devolve *true*. Estamos perante um circuito Hamiltoniano e por conseguinte perante uma solução admissível. Caso exista um vértice com um valor diferente de 3, esta função booleana devolve *false*.

Caso a solução inicial lida pelo MetaHeur não seja admissível, então é sugerido ao utilizador o recurso a uma solução inicial aleatória.

Apresenta-se de seguida o pseudocódigo destes procedimentos:

Procedimento Introdução Solução Inicial Problema TSP

Entradas: Célula superior esquerda da tabela que contém os valores binários da solução inicial a introduzir

Peso, soluçãoinicial(), vértices(), grauvértices() ← 0

Repita Até encontrar célula(linha+i,coluna) vazia

soluçãoinicial(i) ← célula(linha+i,coluna)

Se soluçãoinicial(i) <>0 **e** soluçãoinicial(i) <> 1 **Então**

x ← Mensagem(Valor inválido. Deseja corrigir para 0?)

Se x = Sim **Então**

soluçãoinicial(i) ← 0

Caso Contrário

Soluçãoinicial(i) ← 1

i ← i + 1

Se Tamanho da coluna 1 Tabela <> Tamanho vector soluçãoinicial **Então**

Mensagem(Tamanho incorrecto do vector dos elementos identificadores comparado com o tamanho do vector solução inicial)

Caso Contrário

Para i = 0 **Até** Tamanho vector soluçãoinicial **Faça**

Peso ← Peso + coluna 3 tabela (i) * soluçãoinicial(i)

Próximo i

Para i = 0 **Até** tamanho vector soluçãoinicial **Faça**

Se soluçãoinicial(i) = 1 **Então**

Para l = 0 **Até** tamanho vértices **Faça**

Se coluna 1 tabela (i) = vértices(l) **Então** grauvértices(l) ← grauvértices(l) + 1

Se coluna 2 tabela (i) = vértices(l) **Então** grauvértices(l) ← grauvértices(l) + 2

Se grauvértices (l) for maior do que 3 **Então**

Mensagem(Esta Solução não é admissível. Opte por activar a geração aleatória de uma solução.)

Próximo l

Próximo i

Se CirHam(grauvértices) =False **Então**

Mensagem (Esta Solução não é Admissível. Opte por activar a geração aleatória de uma solução.)

Próxima célula

Fim

Procedimento número vértices (coluna 1 tabela, coluna 2 tabela)

Flg ← False

v ← 0

vértices(0) ← coluna 1 tabela (0)

Para l = 1 **Até** tamanho da coluna 1 tabela **Faça**

flg = False

Para i = 0 **Até** tamanho vértices **Faça**

Se vértices(i) =coluna 1 tabela (l) **Então**

flg ← True

i ← tamanho vértices

Próximo i

Se flg = False **Então**

v ← v + 1

vértices(v) ← coluna 1 tabela (l)

Próximo l

Para l = 0 **Até** tamanho coluna 2 tabela **Faça**

flg ← False

Para i = 0 **Até** tamanho vértices **Faça**

Se vértices(i) = coluna 2 tabela (l) **Então**

flg ← True

i ←tamanho vértices

Próximo i

Se flg =False **Então**

v ← v + 1

vértices(v) ← coluna 2 tabela (l)

Próximo l

Fim

Função CirHam(grauvértices) **Retorna** False ou True

Para i = 0 **Até** tamanho grauvértices **Faça**

Se grauvértices(i) <> 3 **Então**

Retorna False

Próximo i

Retorna ← True

Fim Função

2.3. Geração Aleatória de uma Solução Admissível Inicial

Se o utilizador optar pela geração aleatória da solução inicial do problema, recorre-se a um sorteio dos elementos, no caso do problema do Saco-Mochila, ou dos arcos, no caso do problema do Caixeiro-Viajante, como se detalha nos pontos seguintes.

2.3.1. Geração Aleatória de uma Solução Admissível Inicial para o Problema do Saco-Mochila

Este procedimento vai gerar de forma aleatória uma solução inicial para o problema do Saco-Mochila.

A única indicação necessária para a execução deste procedimento é a capacidade.

O elemento i do vector identificativo dos elementos, possui a agradabilidade definida no vector agradabilidade na posição i e o seu peso corresponde ao valor do vector peso na mesma posição i.

A solução admissível é gerada por sorteio entre os possíveis elementos. Para tal define-se a variável "Limite" que representa o número de elementos que ainda podem ser sorteados. O vector "sorteio", que inicialmente coincide com o vector de identificação dos elementos, é utilizado para a escolha dos elementos a incluir na solução. Recorre-se a números pseudoaleatórios uniformemente distribuídos entre 0 e 1, que multiplicados pelo valor da variável "Limite" e considerando a sua parte inteira identificam os elementos sorteados.

Após a obtenção de um elemento sorteado, elimina-se este elemento do vector sorteio (deixa de poder voltar a sair) e percorre-se todo o vector para que os elementos seguintes ocupem o índice anterior e actualiza-se Limite decrementando o seu valor em uma unidade.

Executa-se este processo repetidamente até que, com a entrada de um novo elemento, se exceda a capacidade. Quando isto acontece, retira-se este elemento que excedeu a capacidade e a

solução admissível inicial poderá ser a obtida pelos outros elementos que não chegaram a exceder a capacidade.

Por razões óbvias, tal como no procedimento anterior, quando se calcula o peso, também se calcula a agradabilidade.

O critério de paragem escolhido não causa problemas de segurança, uma vez que o procedimento vai parar a sua execução com uma mensagem de erro, se forem sorteados todos os elementos e aparecer um índice negativo no vector "sorteio".

Procedimento Gerar aleatoriamente solução inicial problema Saco-Mochila

Entradas: Capacidade

Agradabilidade, Peso, soluçãoinicial() ← 0

Limite ← tamanho do vector coluna 1 da tabela

Para i ←0 **Até** Limite **Faça**

Sorteio(i) ← i

Próximo i

Repita

Sorte ← Inteiro (valor pseudoaleatório entre 0 e 1 * Limite)

Se soluçãoinicial(Sorteio(Sorte)) = 0 **Então**

Soluçãoinicial(sorteio(sorte)) ← 1

Agradabilidade ← Agradabilidade + Coluna 2 da tabela (sorteio(sorte))

Peso ← Peso + Coluna 3 da tabela (sorteio(sorte))

Últimosorteado ← sorte

Últimoagradabilidade ← Coluna 2 da tabela (sorteio(sorte))

Últimopeso ← Coluna 3 da tabela (sorteio(sorte))

Para l ← sorte **Até** Limite – 1 **Faça**

Sorteio(l) ← sorteio(l+1)

Próximo l

Limite ← Limite – 1

Até Peso > Capacidade

Soluçãoinicial(sorteio(sorte)) ←0

Agradabilidade ← Agradabilidade - Coluna 2 da tabela (sorteio(sorte))

Peso ← Peso - Coluna 3 da tabela (sorteio(sorte))

Fim

2.3.2. Geração Aleatória de uma Solução Admissível Inicial para o Problema do Caixeiro-Viajante

Este procedimento vai gerar de forma aleatória uma solução inicial para o problema do Caixeiro-Viajante.

Neste caso, não há nenhum argumento a ser passado para este procedimento e por essa razão não há nenhuma entrada, isto é, apenas se utilizam os dados previamente lidos.

Inicializam-se as variáveis que definem o número dos arcos lidos e as auxiliares para a execução do procedimento.

Tal como no procedimento anterior, cria-se um vector sorteio com os arcos que estão disponíveis para serem incluídas na solução, que será constituído pelos índices que identificam os arcos. Por exemplo, o arco i corresponde ao índice i do vector da coluna 1. De forma idêntica, a variável "Limite" define o tamanho do vector sorteio.

No vector sorteio, como referido, só estão disponíveis os arcos que podem ser considerados. Para o efeito, gera-se um número pseudoaleatório, "sorte", no intervalo entre 0 e o tamanho do vector

sorteio que contém o índice que identifica os arcos. Note-se que o vector sorteio tem os índices que identificam os arcos e a variável "sorte", identifica o arco sorteado.

Após sorteado o arco verifica-se qual o vértice inicial e qual o vértice final e, com recurso ao vector "con", garante-se que não sejam incluídos arcos não admissíveis. Assim, o vector "con" tem o mesmo tamanho que o vector "vértices" e quando se ligam dois vértices, através de um arco, a essa ligação faz-se corresponder um número maior que zero em "con". O maior valor deste vector representa um majorante para o número de componentes conexas na solução. Quando é sorteada uma ligação, (x,y), incidente num vértice, x, já na solução, ou seja, num vértice com valor de "con" positivo, o valor de "con" do outro vértice, y, passa a ser igual a "con(x)". No caso de ser sorteada um novo arco, (x,y), entre dois vértices que não estão na solução, con(x)=con(y)=0, a essa ligação faz-se corresponder um novo número de "con", resultando con(x)=con(y)= 1 + número de componentes conexas já identificadas. Assim, quando a solução é admissível, o vector "con" possui todos os valores iguais uma vez que a mesma conexão liga todos os vértices. No caso de haver valores diferentes, significa que o grafo é desconexo e por conseguinte não existe nenhum circuito Hamiltoniano. Note-se que por esta razão, o sorteio dos arcos é condicionado aos que em cada iteração podem acrescentar a actual solução. Naturalmente que o último arco nunca é sorteado, nesse momento, só irá existir uma possibilidade.

Este procedimento recorre a algumas funções e procedimentos anteriormente descritos, como por exemplo, o "grauvértices" e, por conseguinte, já não serão analisados.

Ao longo deste procedimento são efectuados alguns testes de controlo para garantir que eventuais problemas possam ser detectados atempadamente e a execução do MetaHeur seja parada e o utilizador notificado.

Este procedimento só pára a sua execução quando surge uma mensagem de erro ou a função "CirHam" devolver o valor *true*, ou seja, se se estiver perante um circuito Hamiltoniano representando uma solução admissível.

Procedimento Geração aleatória de solução inicial para o TSP

Entradas:

lmtar ← tamanho coluna 1 tabela ()

soluçãoinicial(),sorteio(), grausvértices(),con(), p, q, ct ← 0

Para i = 0 **Até** lmtar **Faça**

sorteio(i) ← i

Próximo i

Limite ← lmtar

Repete

p ← p + 1

Repete

Sorte ← Inteiro (valor aleatório entre 0 e 1 * Limite)

Para l ← 0 **Até** tamanho vértices **Faça**

Se coluna 1 tabela (sorteio(sorte)) = vértices(l) **Então** vi ← l

Se coluna 2 tabela (sorteio(sorte)) = vértices(l) **Então** vf ← l

Próximo l

Até (con(vi) <>con(vf)) **ou** (con(vi) = con(vf) = 0)

Se 0<con(vf) < con(vi) **Então**

Troca ← con(vi)

Para l ← 0 **Até** tamanho vértices **Faça**

Se con(l) = Troca **Então** con(l) ←con(vf)

Próximo l

Caso Contrário Se 0 < con(vi) < con(vf) **Então**

Troca ← con(vf)

Para l ← 0 **Até** tamanho vértices **Faça**

Se con(l) = Troca **Então** con(l) ← com(vi)

Próximo l

Caso Contrário Se con(vi) = 0 **e** con(vf) >0 **Então**

con(vi) ← con(vf)

Caso Contrário Se con(vf) = 0 **e** con(vi) > 0 **Então**

con(vf) ← con(vi)

Caso Contrário

con(vi) ← p

con(vf) ← p

noini ← coluna 1 tabela (sorteio(sorte))

nofim ← coluna 2 tabela (sorteio(sorte))

Se soluçãoinicial(sorteio(sorte)) = 0 **Então**

soluçãoinicial(sorteio(sorte)) ← 1

grauvértices(vi) ← grauvértices(vi) + 1

grauvértices(vf) ← grauvértices(vf) + 2

Se grauvértices(vi) > 3 **ou** grauvértices(vf) > 3 **Então**

Mensagem (Problema com um grau do vértice vértices(l))

Para l = sorte **Até** Limite – 1 **Faça** sorteio(l) ← sorteio(l + 1)

Limite ← Limite – 1

k ← -1

ct ← 1

Repete

k ← k + 1

Se (coluna 1 tabela (sorteio(k)) = noini) **ou** (coluna 2 tabela (sorteio(k)) = nofim) **ou** ((coluna 1 tabela (sorteio(k)) = nofim) **e** (coluna 2 tabela (sorteio(k)) = noini)) **Então**

ct ← ct + 1

Para l = k **Até** Limite – 1 **Faça**

sorteio(l) ← sorteio(l + 1)

Próximo l

Limite ← Limite - 1

k ← k - 1

Até k = Limite

Se Limite= 0 **Então**

soluçãoinicial(sorteio(0)) = 1

Para l = 0 **Até** tamanho vértices **Faça**

Se coluna 1 tabela (sorteio(0)) = vértices(l) **Então**

grauvértices(l) ← grauvértices(l) + 1

Se coluna 2 tabela (sorteio(0)) = vértices(l) **Então**

grauvértices(l) ← grauvértices(l) + 2

Se grauvértices(l) > 3 **Então**

Mensagem (Este Problema não tem arcos que permitam encontrar uma solução admissível.)

Próximo l

Até CirHam(grauvértices) = True

Fim

3. METAHEURÍSTICA GRASP

3.1. Introdução

Uma das metaheurísticas implementadas é a GRASP-Greedy Randomized Adaptive Search Procedure (Feo e Resende, 1995) que se caracteriza por várias iterações de um procedimento com duas fases. Na primeira fase, executa-se uma heurística construtiva aleatorizada para obter uma solução inicial e, na segunda fase, uma heurística melhorativa numa vizinhança da solução obtida. A melhor solução encontrada é guardada e o procedimento de duas fases será repetido um número previamente especificado de iterações.

Será descrita em pseudocódigo esta metaheurística de uma forma mais genérica para o problema do Saco-Mochila. Aumenta-se o seu detalhe no caso do problema do Caixeiro-Viajante, pois envolve a criação de mais algumas funções e procedimentos.

3.2. GRASP para o Problema do Saco-Mochila

Este procedimento lê, no painel do MetaHeur referente ao problema do Saco-Mochila, Figura 2, os parâmetros deste algoritmo e os respectivos critérios de paragem. Os valores apresentados são os que por defeito são propostos pelo MetaHeur, ficando a cargo do utilizador a decisão de os modificar. Estes critérios de paragem resultam do número máximo de iterações, do número máximo de iterações sem melhorar a solução, do tempo máximo que o algoritmo pode utilizar na sua execução e do tempo máximo que pode utilizar sem melhorar uma solução obtida.

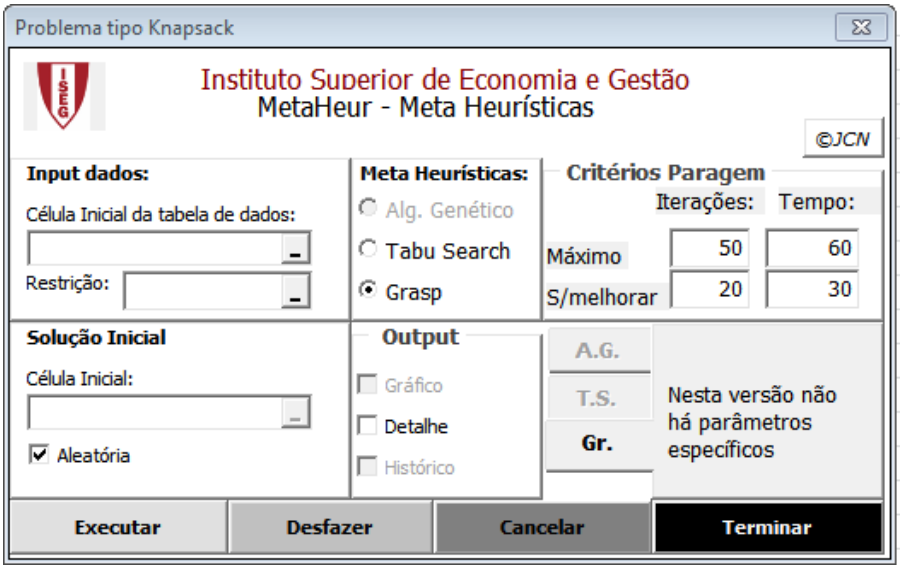


FIGURA 2 - PAINEL SACO-MOCHILA

Para controlar o tempo de execução do programa utiliza-se uma variável "t", que é a diferença entre o instante corrente e o inicial. O valor de início deverá ser muito grande para não parar a execução do primeiro ciclo.

Gera-se aleatoriamente uma solução inicial, de acordo com o procedimento "Geração Aleatória de uma Solução Admissível Inicial" já analisado, e com base nesta solução será aplicada uma heurística melhorativa. Estes dois passos são executados em cada iteração da heurística GRASP.

Na fase melhorativa, é definida uma vizinhança da solução corrente. Considera-se que uma solução é vizinha de uma solução corrente se um dos seus elementos tem valores contrários. A solução vizinha da actual, calcula-se gerando um valor "sorte", correspondente à parte inteira do produto do "tamanho solução" por um número pseudoaleatório entre 0 e 1. Obtido o índice a alterar, basta então modificar o seu valor, de 0 para 1 ou vice-versa. A estratégia de pesquisa na vizinhança é do tipo "best improvement" que consiste em pesquisar toda a vizinhança até encontrar a melhor solução. Como esta estratégia pode consumir muitos recursos computacionais utilizou-se um critério de paragem. É gerado um número pseudoaleatório uniformemente distribuído entre 0 e 1 e, caso seja inferior a 0,1, pára-se a pesquisa da actual vizinhança. Passa-se então para uma nova iteração da GRASP, desde que os critérios de paragem não estejam satisfeitos.

Por facilidade de execução as funções "custo" e "avaliação", retornam o peso e a agradabilidade da nova solução na vizinhança. Caso o "custo" devolva um valor não admissível, procura-se uma outra solução na vizinhança da última solução admissível obtida.

Estas duas funções agora introduzidas ("avaliação" e "custo"), poderiam fazer alterar o pseudocódigo do Cap. 2, para que fossem executadas em vez do pseudocódigo elaborado nessa altura para o mesmo efeito.

De notar que, na implementação efectuada, a heurística construtiva é totalmente aleatorizada, isto é, são usados os procedimentos descritos na secção 2.3, pelo que não são definidos parâmetros específicos para a GRASP.

Procedimento GRASP para Knapsack

Entradas: Número máximo de iterações, tempo máximo de execução do procedimento, Número máximo de iterações sem melhorar a solução obtida, soluçãoinicial(), Agradabilidade da solução inicial, peso da solução inicial, restrição

maxit ← Número máximo de iterações

maxtem ← Tempo máximo de execução do procedimento

smelit ← Número máximo de iterações sem melhorar a solução obtida

tempoini ← Ler relógio computador

best ← Agradabilidade da solução inicial

S←soluçãoinicial()

t ← 100000000000000

i , k, ct← 0

Repete Enquanto i < maxit **e** k < smelit **e** t < maxtem

Se ct>0 **e** (pseudoalaeatório entre 0 e 1 < 0,1) **Então**

S ← Solução gerada aleatoriamente

i←i+1

V ← Solução na vizinhança da solução S, escolhida aleatoriamente

ct ← ct+ 1

Se avaliação(V) > best **e** Custo(V) <= restrição **Então**

best ← avaliaçao(V)

Soluçãoinicial() ← V()

Caso Contrário

k←k+1

Tempofim ← Ler relógio computador

t ← tempofim – tempoini

Próximo

Fim

Função Avaliação(V()) **Retorna** Agradabilidade

Agradabilidade←0

Para i ← 0 **Até** tamanho vector V **Faça**

Agradabilidade ← Agradabilidade + coluna 2 tabela (i) * V(i)

Próximo i

Retorna Agradabilidade

Função Custo(vector()) **Retorna** Peso

Peso←0

Para i ← 0 **Até** tamanho vector V **Faça**

Peso ← Peso + coluna 3 tabela (i) * V(i)

Próximo i

Retorna Peso

3.3. GRASP para o Problema do Caixeiro-Viajante

Este procedimento será um pouco mais detalhado, devido ao facto de não ser tão intuitivo como o anterior. Utiliza os mesmos parâmetros e critérios de paragem definidos para o problema do Saco-Mochila.

Da mesma forma que no problema do Saco-Mochila, em cada iteração é gerada aleatoriamente uma nova solução com recurso ao procedimento "Geração Aleatória de uma Solução Admissível Inicial para TSP", já analisado anteriormente.

A escolha de uma solução na vizinhança da solução actual é feita como se descreve de seguida. São identificados aleatoriamente dois vértices, x e y, da solução inicial. De seguida retiram-se todos os arcos incidentes nestes vértices.

Se os vértices x e y são vértices adjacentes na solução corrente, então substituem-se três arcos e muda-se o sentido actual do arco(x,y), liga-se x com o percursor de y e o percursor de x com y (ver Figura 3).

Se x e y não são vértices adjacentes na solução corrente, então substituem-se os quatro arcos neles incidentes. Os percursores dos vértices x e y são trocados entre si, bem como os seus sucessores. Ou seja, por exemplo, (x,1) passa a (y,1) e (2,y) passa a (2,x) (ver Figura 4).

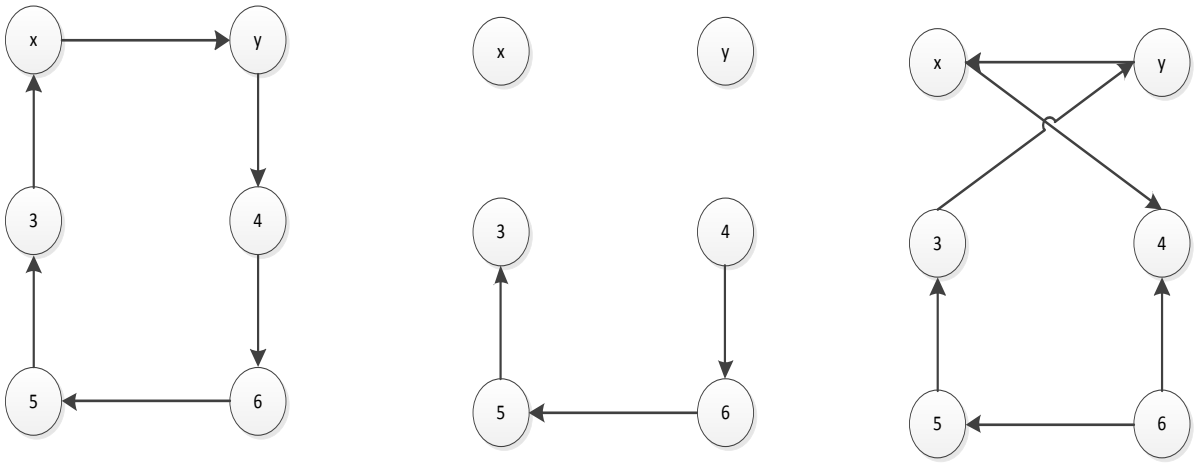


FIGURA 3 - VIZINHANÇA SA DO TSP NÓS SELECCIONADOS CONTÍGUOS

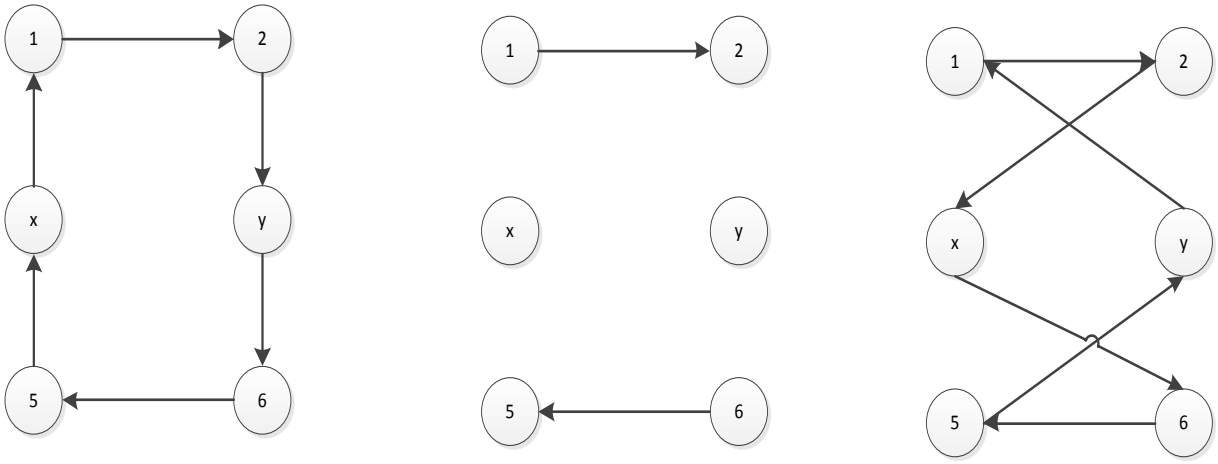


FIGURA 4 - VIZINHANÇA DO TSP NÓS SELECCIONADOS NÃO CONTÍGUOS

Serão efectuadas sempre pelo menos seis ou oito manipulações de arcos, correspondentes à eliminação dos retirados com os acrescentados.

Desta forma, é garantido que a nova solução é admissível pois se a solução inicial era um circuito Hamiltoniano a nova solução também o continua a ser.

Para esta manipulação de arcos foram definidas duas funções "origem" e duas "destino". Propositadamente foram escolhidos nomes diferentes nos parâmetros destes procedimentos. Deve-se estar atento a este facto, quando forem analisados.

Tal como no procedimento anterior, são utilizados pequenos testes para controlar a possibilidade de existência de erros.

O procedimento de escrita da solução obtida no Excel, não parece ter a relevância suficiente para que seja descrito em pseudocódigo. Pode ser consultada a sua codificação directamente no programa, uma vez que o código se encontra disponível.

Quando a nova solução obtida é melhor, tendo em consideração o objectivo, essa solução é guardada através do procedimento "guardabest" e passa a ser esta a solução que se tentará melhorar. Caso contrário, através do procedimento "repõe" volta-se a repor a solução anterior que era até agora a melhor, tendo em consideração o objectivo do problema.

Procedimento GRASP para TSP

Entradas: Número máximo de iterações, tempo máximo de execução do procedimento, Número máximo de iterações sem melhorar a solução obtida, soluçãoinicial(), peso da solução inicial.

maxit ← Número máximo de iterações

maxtem ← Tempo máximo de execução do procedimento

smelit ← Número máximo de iterações sem melhorar a solução obtida

iter← 0

sit←0

tempoini ← Ler relógio do computador

bestsoluçãoinicial() ← soluçãoinicial()

best ← Custo(soluçãoinicial())

k← 0

Repete

iter ← iter + 1

sit ← sit + 1

soluçãoinicial() ← Solução gerada conforme procedimento "Gerar SA Inicial no TSP" (já analisado)

Repete

sort1 ← Inteiro (aleatório entre 0 e 1 * tamanho vértices)

sort2 ← Inteiro (aleatório entre 0 e 1 * tamanho vértices)

Até sort1 <> sort2

no1 ← vértices(sort1)

no2 ← vértices(sort2)

orig1 ← origem(no1, soluçãoinicial(),coluna 1 tabela,coluna 2 tabela())

orig2 ← origem(no2, soluçãoinicial(),coluna 1 tabela,coluna 2 tabela())

dest1 ← destino(no1, soluçãoinicial(),coluna 1 tabela,coluna 2 tabela())

dest2 ← destino(no2, soluçãoinicial(),coluna 1 tabela,coluna 2 tabela())

ct ← 0

j ← -1

Repete

j ← j + 1

Se soluçãoinicial(j) = 1 **Então**

Se coluna 1 tabela (j) = no1 **ou** coluna 2 tabela (j) = no1 **ou** coluna 1

tabela (j) = no2 **ou** coluna 2 tabela (j) = no2 **Então**

ct ← ct + 1

soluçãoinicial(j) ← 0

Peso ← Peso – coluna 3 tabela(j)

Caso Contrário

Se (coluna 1 tabela (j) = orig2 **e** coluna 2 tabela (j) = no1) **ou** (coluna 1 tabela (j) = orig1 **e** coluna 2 tabela (j) = no2) **ou** (coluna 1 tabela (j) = no1 **e** coluna 2 tabela (j) = dest2) **ou** (coluna 1 tabela (j) = no2 **e** coluna 2 tabela (j) = dest1) **Então**

soluçãoinicial(j)← 1

ct ← ct + 1

Peso ← Peso + coluna 3 tabela (j)

Até ct = 8 **ou** j=tamanho coluna 1

Se ct < 6 **Então**

Mensagem(solução não admissível)

Vai para **Fim**.

Se Peso < best **Então**

escreve soluçãoinicial()

guardabest(soluçãoinicial(), bestsolução())

Caso Contrário

k← k+1

tempofim ← Ler relógio computador

t ← tempoini - tempofim

Até i > maxit **ou** k > smelit **ou** t > maxtem **ou** t < smeltem

Fim

Função destino(no, so(), coluna 1 (), coluna 2 ()) **Retorna** destino

Para z = 0 **Até** tamanho coluna 1 **Faça**

Se so(z) = 1 **Então**

Para j = 0 **Até** tamanho coluna 1 **Faça**

Se coluna 1 (j) = no **Então**

destino ← coluna 2(j)

Retorna destino

Próximo j

Mensagem(Destino não alcançado. Repõe solução anterior)

Próximo z

Fim Função

Função origem(no, so(), coluna 1 (), coluna 2 ()) **Retorna** origem

Para z =0 **Até** tamanho coluna 1 **Faça**

Se so(z) = 1 **Então**

Para j = 0 **Até** tamanho coluna 1 **Faça**

Se coluna 2 (j) = no **Então**

origem ← coluna 1(j)

Retorna origem

Próximo j

Mensagem(Destino não alcançado. Repõe solução anterior)

Próximo z

Fim Função

Procedimento guardabest(so(), bso())

Para l ← 0 **Até** tamanho so **Faça**

bso(l) ← so(l)

Próximo i

Fim

4. METAHEURÍSTICA TABU SEARCH

A metaheurística Tabu-Search (Glover,1989;+Hillier e Liberman, 2010) baseia-se em pesquisa local, no entanto, um movimento de uma solução corrente para outra numa sua vizinhança pode ser aceite mesmo não sendo melhorativo.

Com esta possibilidade corre-se o risco de voltar a analisar soluções já analisadas, pelo que há a necessidade da definição de memória de curto prazo, através das chamadas listas tabu. Por outro lado, recorre-se à diversificação para poder efectuar buscas em regiões fora da vizinhança da solução corrente, há muito tempo não exploradas.

A definição de vizinhança e a estratégia de pesquisa coincide com as usadas na GRASP para os dois problemas.

Neste ponto é detalhada a metaheurística Tabu Search para o problema do Saco-Mochila.

A sua adaptação ao problema do Caixeiro-Viajante resume-se apenas à introdução das alterações exibidas e descritas ao longo do Cap. 2 e do Cap. 3. Tratando-se de uma implementação semelhante optou-se por não a descrever. Qualquer questão poderá ser analisada directamente no código.

Para evitar que o pseudocódigo seja demasiado extenso e haver o perigo de se tornar excessivamente confuso, serão agrupadas algumas rotinas menos importantes. No entanto, há sempre a possibilidade de se recorrer ao código da aplicação para se analisar a implementação deste algoritmo.

Ao ser escolhida esta Metaheurística é apresentado ao utilizador o painel da Figura 5, já com os parâmetros e o critério de paragem adequados.

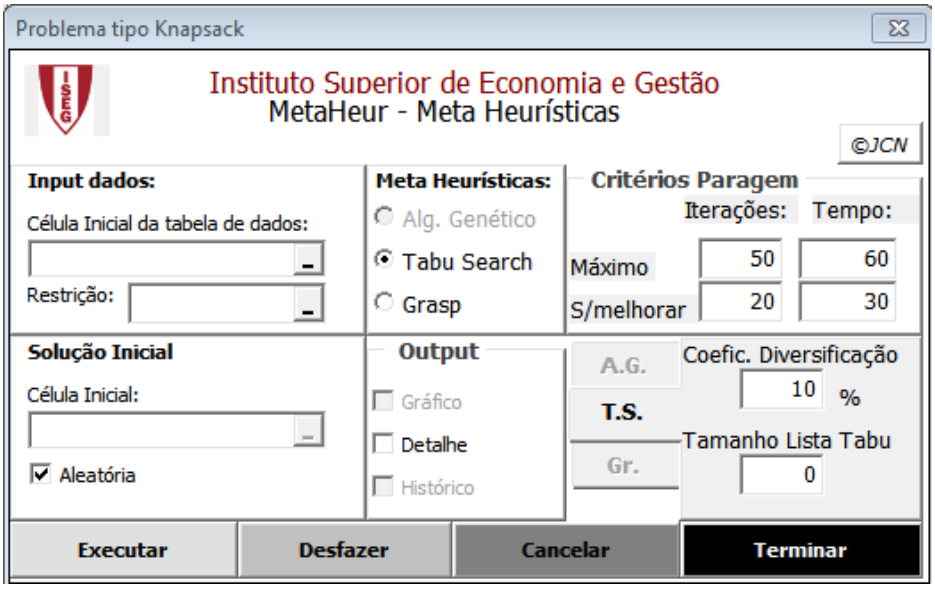


FIGURA 5 - PAINEL SACO-MOCHILA PARA TABU-SEARCH

Os critérios de paragem são exactamente os mesmos que foram definidos para o algoritmo GRASP e, como tal, detalhados no capítulo anterior. Contudo, ao seleccionar o algoritmo Tabu Search, surgem dois parâmetros novos: o coeficiente de diversificação e o tamanho da lista tabu. O coeficiente de diversificação estabelece a probabilidade com que se pretende procurar soluções fora da vizinhança, tendo-se fixado, por defeito o valor de 10%. Esta escolha baseou-se em testes preliminares efectuados, embora versando poucos elementos. O tamanho da lista tabu identifica o número de movimentos que não são analisados durante um certo número de iterações. Os movimentos que não se pretendem analisar são classificados de tabu e colocados na lista. Por defeito considerou-se que o seu tamanho é zero, o que significa que não tem memória. Note-se que este parâmetro deve estar relacionado com o tamanho dos elementos identificativos do problema do Saco-Mochila. Tal como os restantes parâmetros, também o coeficiente de diversificação e o tamanho da lista tabu, podem ser alterados pelo utilizador.

Neste algoritmo são contabilizadas as frequências com que os elementos vão sendo sorteados para incluir na solução. A escolha cai sempre no elemento com menor frequência. A diversificação actua quando se gera um número pseudoaleatório, uniformemente distribuído entre 0 e 1, inferior à taxa de diversificação. Neste caso é sorteado um elemento de entre todos os possíveis, utilizando-se como recurso a função de geração de números pseudoaleatórios tão referenciada ao longo deste trabalho.

O elemento seleccionado é incluído ou excluído da solução conforme não lhe pertença ou esteja nela incluído, respectivamente, e se possível, ou seja, se compatível com a capacidade. Recorre-se à função "custo", já detalhada em capítulos anteriores, para verificar se a nova solução é admissível.

Assim, com a nova solução admissível basta recorrer à função "avaliação" (também já analisada anteriormente) e verificar se esta nova solução é melhor do que a anterior, considerando o objectivo pretendido. Caso o seja, é guardada esta nova solução, se não é reposta a solução anterior, prosseguindo-se sempre com um novo elemento.

Se o tamanho da lista tabu for positivo, o elemento seleccionado quer por sorteio, quer por ter a frequência mais baixa, é guardado na lista "tabu" e não pode voltar a ser seleccionado enquanto constar desse vector. Sempre que a lista já esteja cheia, o elemento mais antigo é eliminado para poder receber o novo elemento seleccionado.

Repete-se o procedimento até um dos critérios de paragem se ter verificado.

É apenas resumido o pseudocódigo deste procedimento, de forma a se poder rapidamente verificar o seu funcionamento. No caso de haver algum interesse na forma como pode ser implementado, pode-se consultar o código no CD em anexo.

Procedimento Tabu Search para o Knapsack

Entradas: Número máximo de iterações, tempo máximo de execução do procedimento, Número máximo de iterações sem melhorar a solução obtida, soluçãoinicial(), Agradabilidade da solução inicial, peso da solução inicial, restrição, coeficiente de diversificação

maxit ← Número máximo de iterações

maxtem ← Tempo máximo de execução do procedimento

smelit ← Número máximo de iterações sem melhorar a solução obtida

tempoini ← Ler relógio computador

best ← Agradabilidade da solução inicial

div ←Coeficiente de diversificação

frequências()←0

p ← 0

q ← 0

S ← Soluçãoinicial()

tempoini ← Ler relógio computador

Repete Até p > maxit **ou** q > smelit **ou** t > maxtem

p ← p + 1

Se (valor aleatório entre 0 e 1) > div **Então**

V ← Escolhe vizinho admissível com elemento x de menor frequência

frequência(x) ← frequência(x) +1

Se Agradabilidade(V) > Agradabilidade (S) **Então**

S ← V

q ← 0

Tabu ← x

Caso Contrário

Tabu ← x

q ← q+1

Caso Contrário

x ← Inteiro(valor aleatório entre 0 e 1* tamanho coluna 1 tabela)

frequência(x) ← frequência(x) +1

V ← Escolhe vizinho admissível

Se x ∈ Tabu **ou** Agradabilidade(V) < Agradabilidade(S) **Então**

q ← q+1

Tabu ← x

Caso Contrário

Se Agradabilidade(V) > Agradabilidade(S) **Então**

S ← V

q ← 0

Tabu ← x

Caso Contrário

Tabu ← x

q ← q + 1

tempofim ← Ler relógio computador

Elimina elemento mais antigo no vector Tabu

t ← tempofim – tempoini

Próximo

Escreve informação sobre tempos de execução, número de iterações e outros indicadores

Fim

Neste procedimento, devidamente resumido para que fique mais claro o que se vai fazer, não é feita qualquer referência à forma como se manipula e ordena a frequência. Pela sua importância neste algoritmo, será analisado.

Tratando-se de um trabalho em que as motivações são mais viradas para um ponto de vista académico, optámos por usar o algoritmo *Bubble Sort*, para uma matriz com 2 colunas. Para o efeito criou-se um procedimento que faz passar por argumento a matriz frequências constituída pela identificação dos arcos na coluna 0 e pela respectiva frequência na coluna 1 e como outro argumento a coluna que é o critério de ordenação, que neste caso seria a coluna 1.

Procedimento BubbleSort (frequência() , 1)

SortColuna ← 1

```
Para i = 0 Até tamanho(frequência, 1) – 1 Faça
    Para j = 0) Até tamanho(frequência, 1) – 1 Faça
        Se frequência(j, SortColuna) > frequência(j + 1, SortColuna) Então
            Para y = 0 Até tamanho(frequência, 2) Faça
                t ← frequência(j, y)
                frequência(j, y) ← frequência(j + 1, y)
                frequência (j + 1, y) ← t
            Próximo y
    Próximo j
Próximo i

Fim
```

Para melhorar o algoritmo, bastará criar um novo procedimento que substitua este último, por exemplo, um procedimento que utilize o algoritmo *QuickSort3* para efectuar a ordenação da matriz de frequências, por ser dos mais eficazes, o que poderá ser confirmado em http://www.sorting-algorithms.com/quick-sort-3-way. Mas será necessário adaptá-lo para o caso de um array multidimensional, como é o das frequências.

5. ALGORITMO GENÉTICO

As metaheurísticas descritas anteriormente partem de uma solução corrente para análise de outra solução. Os algoritmos genéticos (Goldberg,1989) trabalham com uma "população de soluções (indivíduos) e baseiam-se na analogia entre o processo de pesquisa por melhores soluções e o conceito de sobrevivência dos mais aptos, da Teoria da Evolução das espécies. Soluções menos aptas são substituídas por novas soluções, descendentes de progenitores de boa "qualidade".

Neste capítulo será feita a descrição do pseudocódigo de um algoritmo genético simples apenas para o problema do Saco-Mochila, tal como na metaheurística Tabu-Search. A sua adaptação para o problema do Caixeiro-Viajante resume-se à introdução das alterações exibidas ao longo do Cap. 2 e do Cap. 3.

Este algoritmo não chegou a ser implementado no MetaHeur, mas está pré-preparado para o ser.

Para evitar que o pseudocódigo seja demasiado extenso e haver o perigo de se tornar excessivamente confuso, serão agrupadas algumas rotinas menos importantes.

Neste algoritmo torna-se necessário definir previamente uma nomenclatura e só posteriormente a sua implementação, propriamente dita.

5.1. Nomenclatura:

Apresentam-se de seguida os principais conceitos subjacentes ao algoritmo genético que se pretende implementar:

Cromossoma: Representa uma solução do problema. Trata-se de um vector binário, constituído pelo valor 1 se o elemento faz parte da solução, ou pelo valor 0, no caso desse elemento não ser seleccionado.

Locus: Representa a posição da variável no cromossoma, ou seja, o índice do vector binário.

Gene: Representa a variável num determinado locus, no caso representa o elemento candidato.

Alelo: Representa o valor do gene, isto é, 1 se é seleccionado e integra a solução e 0 caso não integre a referida solução.

Aptidão: Representa, neste caso, a função objectivo que se pretende maximizar. Define-se desta forma a qualidade dos cromossomas. Poder-se-ia retirar à função o seu valor mínimo e assim, apenas trabalhar com os valores relativos. Não foi feito para manter a coerência para com outros tipos de resoluções efectuadas.

Cruzamento: Combinação de 2 cromossomas pais que originam 1 cromossoma filho. Este cruzamento é feito por selecção aleatória do Locus do cromossoma do primeiro pai e preenchido com os genes do primeiro pai até à posição Locus e como o genes do segundo pai da posição Locus+1 até ao fim do cromossoma. Exemplificando, obtido aleatoriamente entre 1 e 15, por exemplo 4 o Locus onde se efectua o cruzamento, os primeiros 4 são os genes do primeiro pai e os 11 restantes serão os genes do segundo pai. A escolha de um ponto de cruzamento faz-se por razões computacionais e para maior rapidez na convergência. Para compensar este facto e aumentar a diversificação, a taxa de mutação vai ser superior ao habitual e a mutação vai ser do tipo total.

Selecção: A selecção dos pais para cruzamento é efectuada pelo método da roleta, onde a probabilidade de ser seleccionado é proporcional ao valor da sua função aptidão. Para avaliação mais detalhada poderá ser consultado o pseudocódigo da função selecção.

Mutação: Mutação Total, com uma probabilidade pequena a mutação altera completamente os valores dos genes do filho, isto é, troca todos os bits do vector filho resultante de um cruzamento.

Taxa de Mutação: Representa a probabilidade com que um filho sofre o efeito mutação. Usualmente considera-se um valor pequeno, cerca de 5%. Contudo, neste trabalho, pretendendo-se proporcionar uma maior diversificação utilizou-se uma taxa de 10%, uma vez que por razões computacionais foi escolhido apenas um ponto para o cruzamento.

Elitismo: Para não se perder a melhor solução gerada obtida até então, inclui-se sempre o melhor cromossoma da geração anterior na geração actual.

População: Foi referido quer nas aulas de Heurísticas, quer na literatura consultada, que a dimensão da população se deverá situar entre o tamanho do cromossoma e o dobro desse tamanho. No entanto, deverá haver, um equilíbrio interessante entre a rapidez computacional e um espaço de soluções suficientes a ser explorado. Por exemplo, no caso de um problema com 15 elementos, a população dever-se-á situar entre os 15 e os 30 elementos, podendo ser usado um valor intermédio do intervalo dado ser considerado uma boa dimensão.

5.2. Algoritmo Genético para o problema do Saco-Mochila

A implementação deste algoritmo está preparada nos painéis criados pelo MetaHeur, mas de momento encontra-se desactivado.

Um dos parâmetros que este algoritmo deve ter é o da dimensão da população. Por defeito, é proposto ao utilizador o arredondamento para o inteiro mais próximo da soma do tamanho do vector de identificação com metade desse tamanho. A dimensão da população poderá estar relacionada com a forma como esta evolui, podendo ser um valor constante. Neste caso, em cada iteração deve ser sorteado um elemento da população para ser eliminado. Outra forma de manter constante o tamanho da população poderia passar pela criação de uma função "retirar" que periodicamente (considerando o tempo decorrido desde o início da execução do algoritmo) eliminaria um elemento da população. Este, pode representar o elemento com menor aptidão ou pode resultar de uma escolha aleatória, que teria o inconveniente de poder eliminar o mais apto.

No pseudocódigo que se apresenta não foi introduzido o controlo da população, pois este resume-se à introdução de um ciclo que analisa, em cada iteração, a função "retirar" e a executa. Alternativamente, pode incluir-se um teste, imediatamente antes dos critérios de paragem, onde em caso da dimensão da população ter atingido o seu valor máximo, se retira o elemento menos apto ou se sorteia um a retirar.

Outra possibilidade mais criativa, passa por considerar que a população é constituída sempre por duas gerações, e por cada nova geração criada elimina-se a mais antiga. Esta ideia é a que se pretende explorar no procedimento que vai ser analisado.

O procedimento inicializa as suas variáveis com a passagem de alguns parâmetros, como por exemplo, a solução inicial, e outros que são lidos directamente do painel do problema, como por exemplo a taxa de mutação.

Inicialmente, constrói-se a população de forma aleatória, gerando valores pseudo aleatórios entre 0 e 1 para os genes de cada cromossoma, em número igual ao tamanho do cromossoma (ou seja, até ao tamanho do vector identificativo dos elementos). Concluída esta geração de genes, o cromossoma é adicionado à população geral. De seguida é então executado o algoritmo genético propriamente dito, com o recurso a uma função que recebe a população (ou melhor a geração anterior) e a aptidão. Na prática a aptidão é uma função que é evocada sempre que necessário.

Esta função retorna o melhor cromossoma, indicando, no final, a solução proposta por este algoritmo para o problema.

A selecção dos pais é feita pela função "selecção" que devolve o pai identificado pelo método da roleta. Neste, gera-se um valor pseudoaleatório uniformemente distribuído entre 0 e 1, que multiplicado pela melhor aptidão total obtida, vai determinar o valor mínimo da aptidão que o cromossoma pai tem de respeitar para ser seleccionado para o cruzamento. Quando os pais são seleccionados, a função cruzamento define a posição do gene no cromossoma que tem os genes de um pai e a partir do qual se colocarão os do outro pai. O cruzamento faz-se pela concatenação, que é uma função disponível nas linguagens de programação para juntar duas cadeias de caracteres.

Sempre que se obtiver um cromossoma que represente uma solução não admissível, a aptidão desse cromossoma será igualada a zero.

Este processo repete-se até um dos critérios de paragem já referidos se verificar.

Procedimento Algoritmo Genético para o Knapsack

Entradas: Número máximo de iterações, tempo máximo de execução do procedimento, Número máximo de iterações sem melhorar a solução obtida, soluçãoinicial(), Agradabilidade da solução inicial, peso da solução inicial, restrição, tamanho população

População inicial ← 0

Para i = 1 **até** tamanho da população **Faça**

Para j = 1 **até** tamanho cromossoma **Faça**

Se (aleatório[0,1] > 0,5 **então** Gene(j)=1

Caso contrario Gene(j)=0

Adiciona Gene(j) ao Cromossoma (i)

Próximo j

Adiciona cromossoma (i) à População inicial

Próximo i

AlgoritmoGenético (População, Aptidão)

FIM

Função AlgoritmoGenético (População, Aptidão) **Retorna** Melhor Cromossoma

Entradas: População (conjunto de cromossomas) e Aptidão (função que mede a aptidão do cromossoma);

Melhor Cromossoma ← MelhorCromossoma (População, Aptidão)

Repita

Nova população ← Conjunto vazio

Adiciona melhor cromossoma à população

Para i ← 1 **até** tamanho da população **faça**

X ← Selecção (População, Aptidão)

Y ← Selecção (População, Aptidão)

Filho ← Cruzamento(X, Y)

Se (pequena probabilidade aleatória) **então**

Mutação (Filho)

Adiciona filho à população

Se Aptidão(filho) > Aptidão (Melhor Cromossoma) **então**

Melhor Cromossoma ← filho

Próximo i

Até 90% da população ter Aptidão semelhante ou após um determinado período

Retorna Melhor cromossoma de acordo com a Aptidão.

Função Cruzamento (X, Y) **Retorna** Cromossoma Filho

n ← Comprimento (X)

c ← aleatório [1,n]

Retorna Concatena (SubCadeia(X,1,c), SubCadeia(Y,c+1,n))

Função Selecção (População, Aptidão) **Retorna** Cromossoma Pai

TotalAptidão ← 0

Para i ← 1 **até** tamanho da população **faça**

TotalAptidão ← TotalAptidão + Aptidão (Cromossoma i)

Próximo i

Limite ← aleatório [0, TotalAptidão]

Soma ← 0

i ← 0

Repete

i ← i + 1

Soma ← Soma + Aptidão (Cromossoma (i)) / TotalAptidão

Até Soma >= Limite

Retorna Cromossoma(i)

Função Mutação (Filho) **Retorna** Filho com Mutação Completa

n ← Comprimento (Filho)

Para i ← 1 **até** n **faça**

Se Gene(i) = 0 **então** Gene(i) =1

Caso contrário Gene(i) = 0

Próximo i

Retorna Filho

Função MelhorCromossoma (População, Aptidão) **Retorna** Melhor Cromossoma

Melhor Cromossoma ← Cromossoma 1

Para i ← 1 **até** tamanho da população **faça**

Se Aptidão(i) > Aptidão(Melhor Cromossoma) **então**

Melhor Cromossoma ← Cromossoma(i)

Próximo i

Retorna Melhor Cromossoma

Função Aptidão (Cromossoma) **Retorna** Soma da Agradabilidade

n ← Comprimento (Filho)

Agradabilidade ← 0

Custos ← 0

Para i ← 1 **até** n **faça**

Se Gene (i) = 1 **então**

Custos ← Custos + Custo(Gene i)

Se Custos superiores à restrição. **então**

Retorna 0

Caso contrário Agradabilidade ← Agradabilidade + Agradabilidade(i)

Próximo i

Retorna Agradabilidade

6. MANUAL

O Metaheur foi desenvolvido em VBA (Green at al, 2007) para o Excel 2010, apesar de se ter iniciado no 2007 e posteriormente se ter feito a sua migração para a versão 32 *bits* do 2010. Desconhece-se o seu comportamento na versão 64 *bits.*

O friso foi desenvolvido com o Visual Studio Premium. Poderá ser efectuado com a versão Express que é gratuita. Foram utilizados os icons pré existentes, por falta de tempo.

Para a análise do seu código ou para o seu desenvolvimento torna-se necessário activar o menu programador do Excel 2010, relembrando que não existe nenhuma *password* a inibir o acesso às fontes do MetaHeur.

O programa é fornecido com o CD que se encontra em anexo.

6.1. Introdução e Pré-requisitos

Este documento destina-se a dar apoio ao manuseamento da aplicação MetaHeur, quer quanto aos pré-requisitos e manual de instalação da aplicação, quer como um manual de utilizador, onde deverão ser explicadas as funcionalidades principais da aplicação.

Esta aplicação permite a execução de algumas metaheurísticas para o problema do Caixeiro-Viajante e para o problema do Saco-Mochila.

Pretende-se também incluir neste manual os procedimentos de instalação e de desinstalação, quer do ponto de vista da execução de um programa desenvolvido para o efeito, quer do ponto de vista de manipulação dos dados.

Às dificuldades próprias deste trabalho, acresceu o facto de a Microsoft efectuar um *upgrade* ao Excel da versão 2007 para 2010, o que levou também a que houvesse a preocupação de incluir instruções detalhadas para a sua instalação para o 2007 e 2010 versão 32 *bits*. A versão de 64 *bits* não foi analisada pois a sua migração traria problemas de compatibilidade com as versões anteriores do Excel.

Os Pré-Requisitos do MetaHeur podem ser sintetizados da seguinte forma:

Sistema Operativo: Poderá correr em qualquer sistema operativo quer de 32 *bits*, quer de 64 *bits*, desde que possua o aplicativo Excel instalado. Apesar do sistema *Mac OSX* reunir estas condições não é executada no Excel 2008, pois o *Visual Basic for Applications* (Green, J., S., Bovery, R e Alexander, M., 2007) não é suportado, enquanto na versão Excel 2004 poderá ser executado mas sem todas as suas funcionalidades. Nas versões *OpenOffice* em sistemas *Linux* não foram efectuados quaisquer testes e é de prever que o programa não seja executado. Nos sistemas *Windows* pode ser usado *XP*, *Vista* ou *Windows 7*. Nas versões anteriores dos sistemas operativos *Windows* não se efectuaram quaisquer testes, pelo que desaconselhamos a sua utilização;

***Software* previamente instalado:** Torna-se necessário que esteja previamente instalado o Excel 2010 versão 32 *bits*, de preferência em Português. Quanto às outras línguas, a aplicação é executada e não são de prever problemas, podendo no entanto apresentar alguma instabilidade, uma vez que o seu desenvolvimento foi efectuado tendo em consideração a versão de Excel 2010 referida. A versão 64 *bits* do Excel 2010 não foi nem será testada no âmbito deste trabalho, pois obrigaria a uma conversão manual do actual código que iniciou o seu desenvolvimento no Excel 2007 e foi migrado para 2010 32 *bits*, onde se concluiu o trabalho. Quanto às versões Excel 2003 e 2007, pode ser executado sem problemas de relevo. Outro tipo de instabilidade que a aplicação poderá apresentar, poder-se-á dever ao controlo da Microsoft que define as *ranges* do Excel para a introdução dos dados, que apresenta ele próprio alguma instabilidade, já documentada pela própria Microsoft, havendo muita informação disponível na net se efectuar uma busca por *refedit problems*, pois o MetaHeur utiliza este controlo. Por vezes basta um fechar e abrir do Excel para que um eventual problema desapareça;

Relativamente ao hardware do seu computador :

Memória: O programa está limitado à capacidade de endereçamento do sistema operativo;

Disco: Em termos de espaço a aplicação tem um tamanho inferior a 100 *kilobytes*.

6.2. Limitações

Não são conhecidas limitações ou constrangimentos directamente imputáveis à aplicação.

Esta está limitada pela capacidade de endereçamento do sistema operativo e pelo *hardware* onde o sistema operativo está instalado.

Se for executado como *add-in* do Excel 2007, terá as limitações deste aplicativo que são as seguintes:

1.048.576 linhas por 16.384 colunas;

Máximo de 32767 caracteres por célula;

O número de execuções está dependente do número de folhas de um livro, ou seja, apenas está limitado pela memória do computador (*Hardware* ou capacidade de endereçamento do sistema operativo).

Pelo tipo de variáveis escolhidas, a aplicação está limitada às seguintes grandezas:

Menor número negativo: -2.147.483.648;

Maior número positivo: 2.147.483.647.

No caso de o executarmos como suplemento do Excel 2010, teremos as mesmas limitações que as referidas, para o Excel 2007 na versão de 32 *bits*.

6.3. Instalação

Tendo como pré-requisito a instalação do Excel, vamos supor que já se encontra instalado.

A aplicação MetaHeur pode ser instalada manualmente, desde que se proceda da seguinte forma:

Windows 7 e Excel 2007:

Mova o ficheiro ***MetaHeur.xlam*** para a seguinte pasta:

C:\Program Files (x86)\Microsoft Office\Office12\Library\

Windows 7 e Excel 2010 versão 32 *bits*:

Mova o ficheiro ***MetaHeur.xlam*** para a seguinte pasta:

C:\Users\[sua conta]\AppData\Roaming\Microsoft\Suplementos

Modo geral:

O ficheiro ***MetaHeur.xlam*** deverá ser copiado para uma pasta onde encontrar uma estrutura de pastas e ficheiros semelhantes à da seguinte figura:

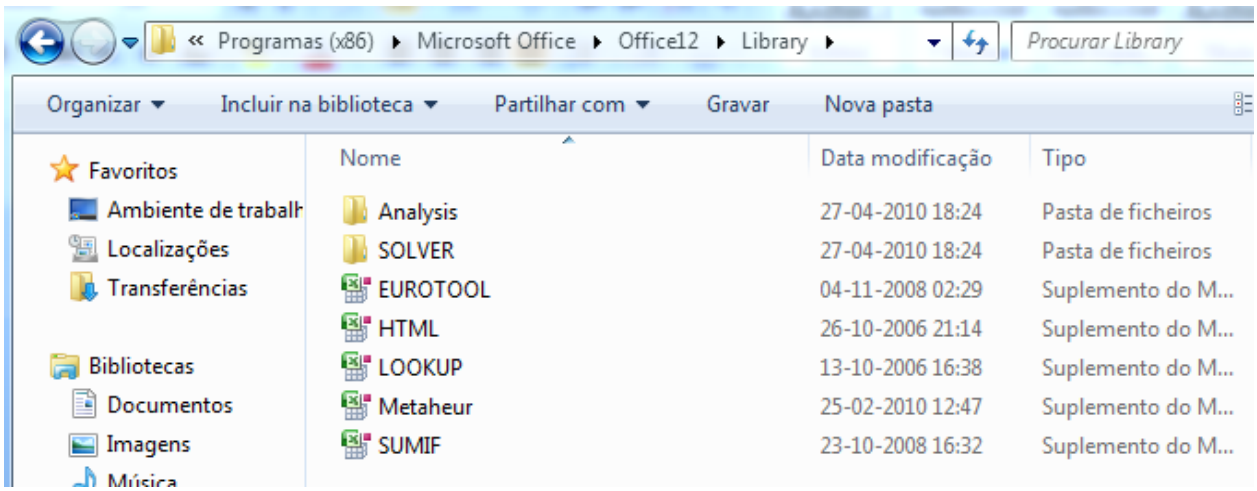


FIGURA 6 - ESTRUTURA SUPLEMENTOS EXCEL

Note que esta aplicação é instalada como o são todos os suplementos (*Add-in*) do Excel.

Assim, colocado o ficheiro que contém a programação no local adequado para os suplementos do Solver, torna-se necessário executar os seguintes passos no Excel:

a. Caso Excel 2007

Efectuar um clique no botão do Office  ;

Dentro do painel exibido pressione o botão de **Opções do Excel**, de acordo com a figura seguinte:

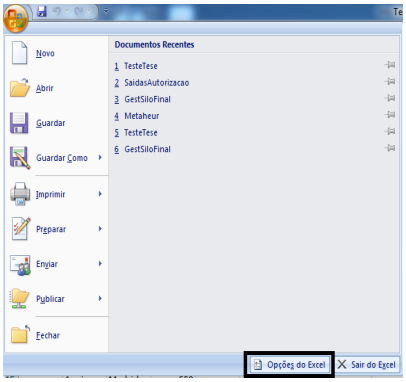


FIGURA 7 - BOTÃO OPÇÕES DO EXCEL

No painel que lhe será apresentado, deverá seleccionar **Suplementos** e pressionar o botão **Ir…** :

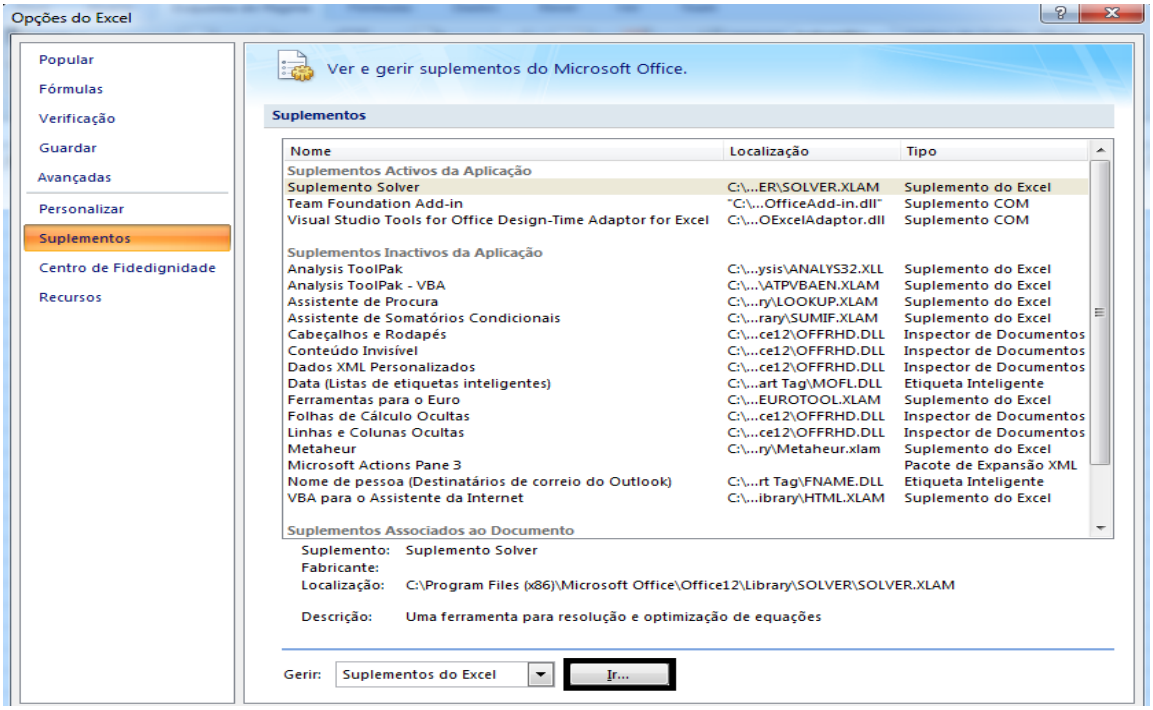


FIGURA 8 - SUPLEMENTOS EXCEL

Deverá finalmente seleccionar o MetaHeur conforme a figura abaixo e pressionar em **Ok**:

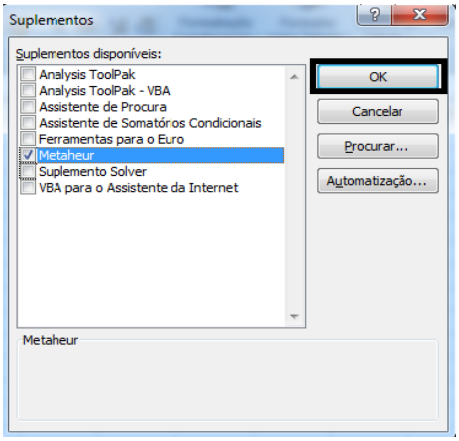


FIGURA 9 - PAINEL DE SUPLEMENTOS INSTALÁVEIS

Concluídos os procedimentos referidos, o seu Excel deverá passar a exibir um novo separador com o nome **MetaHeur**, com 3 opções conforme as figuras abaixo:

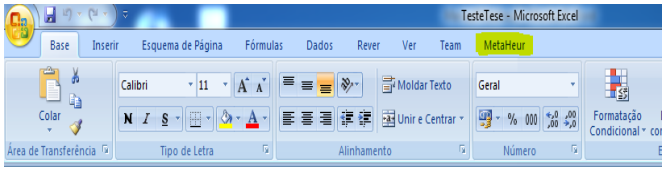


FIGURA 10 - NOVO SEPARADOR

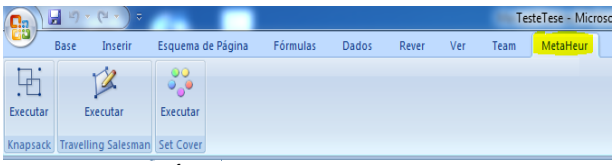


FIGURA 11 - CONTEÚDO DO SEPARADOR

A partir de agora, o MetaHeur está instalado e pronto para ser utilizado.

b. Caso Excel 2010 versão 32 bits

Efectuar um clique no botão Ficheiro 

;

Dentro do painel exibido pressione **Opções**, de acordo com a figura seguinte:

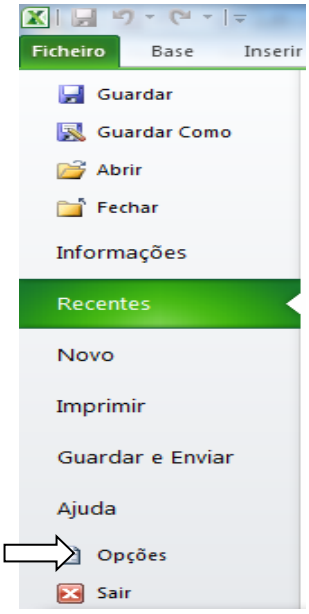


Figura 12 - Opções Excel 2010

No painel que lhe será apresentado, deverá seleccionar **Suplementos** e pressionar o botão **Ir…** :

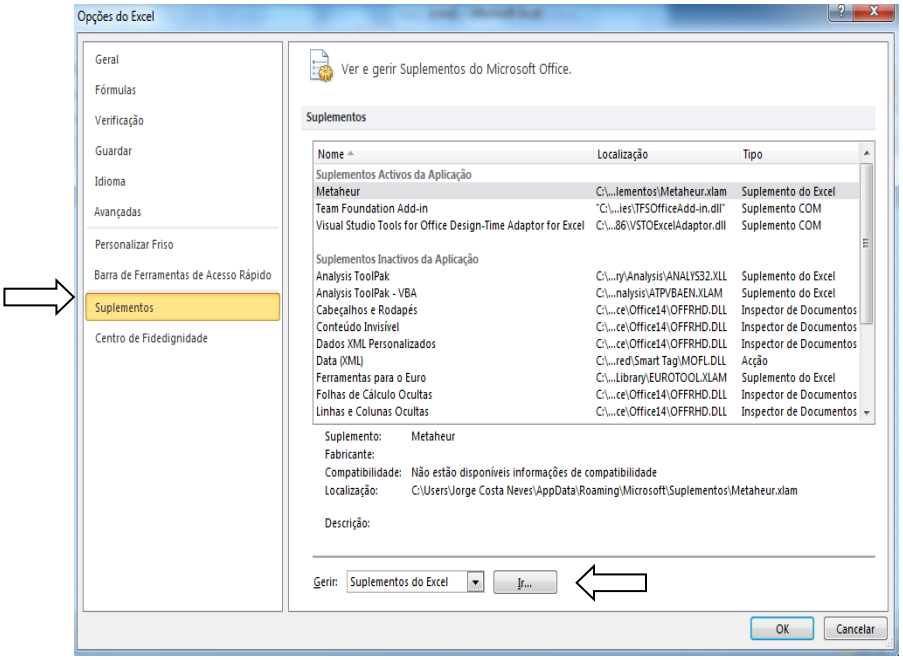


Figura 13 - Suplementos Excel 2010

Deverá finalmente seleccionar o MetaHeur conforme a figura abaixo e pressionar em **Ok**:

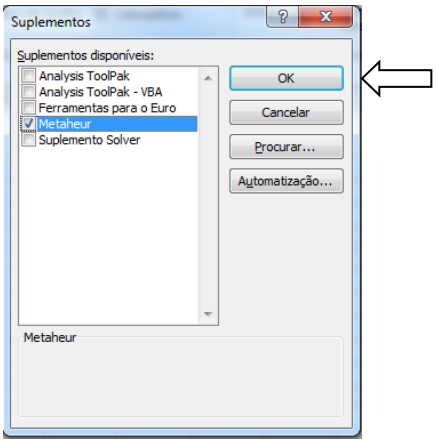


Figura 14 - Painel de Suplementos instaláveis no Excel 2010

Concluídos os procedimentos referidos, o seu Excel 2010 deverá passar a exibir um novo separador com o nome **MetaHeur**, com 3 opções conforme as figuras abaixo:

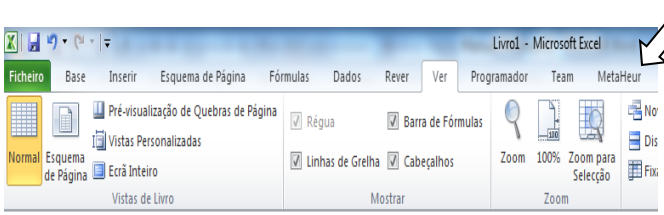


Figura 15 - Novo separador

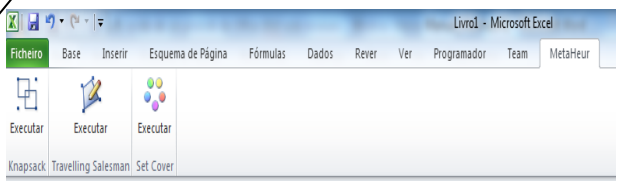


Figura 16 - Conteúdo do separador

A partir de agora, o MetaHeur está instalado e pronto para ser utilizado.

6.4. Pressupostos

O programa pressupõe o conhecimento dos tipos de problemas abordados: Saco-Mochila e Caixeiro-Viajante.

Também se considera que o utilizador tem um conhecimento básico da aplicação MS Excel e dos princípios de manuseamento de ficheiros dos sistemas operativos *Windows.*

6.5. Friso MetaHeur

O friso do MetaHeur apresenta-se dividido de acordo com o tipo de problema que se pretende solucionar, como se ilustra na Figura 17

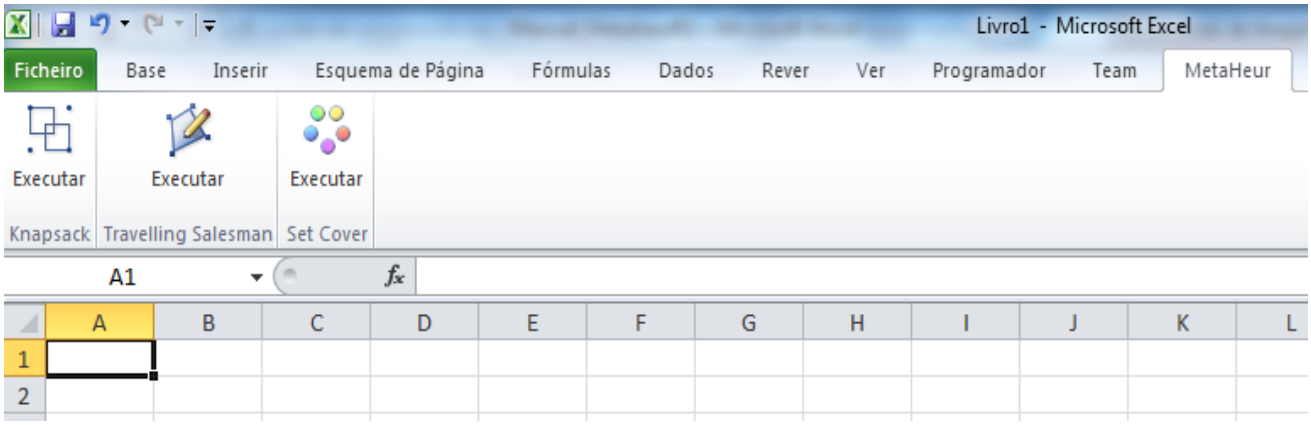


Figura 17 - Friso metaheur

Premindo o botão do separador correspondente ao tipo de problema, por exemplo "*Set Cover*", resultará na exibição de um painel que exiba os parâmetros correspondentes ao tipo de problema. No caso específico, como ainda não está disponível será exibida a seguinte mensagem:

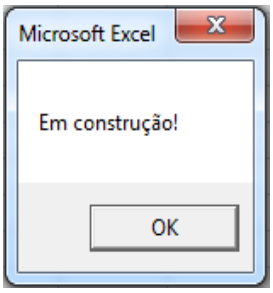


Figura 18 - Painel problema *set cover*

Qualquer dos outros separadores estão operacionais e por conseguinte os painéis correspondentes a cada tipo de problema será detalhadamente abordado nos capítulos seguintes.

6.6. Problema do Saco-*Mochila* (Knapsack)

Caso se pretenda resolver um problema de Saco-Mochila, deve pressionar-se o botão correspondente que se encontra no separador *Knapsack* do friso do MetaHeur:

É exibido o seguinte painel:

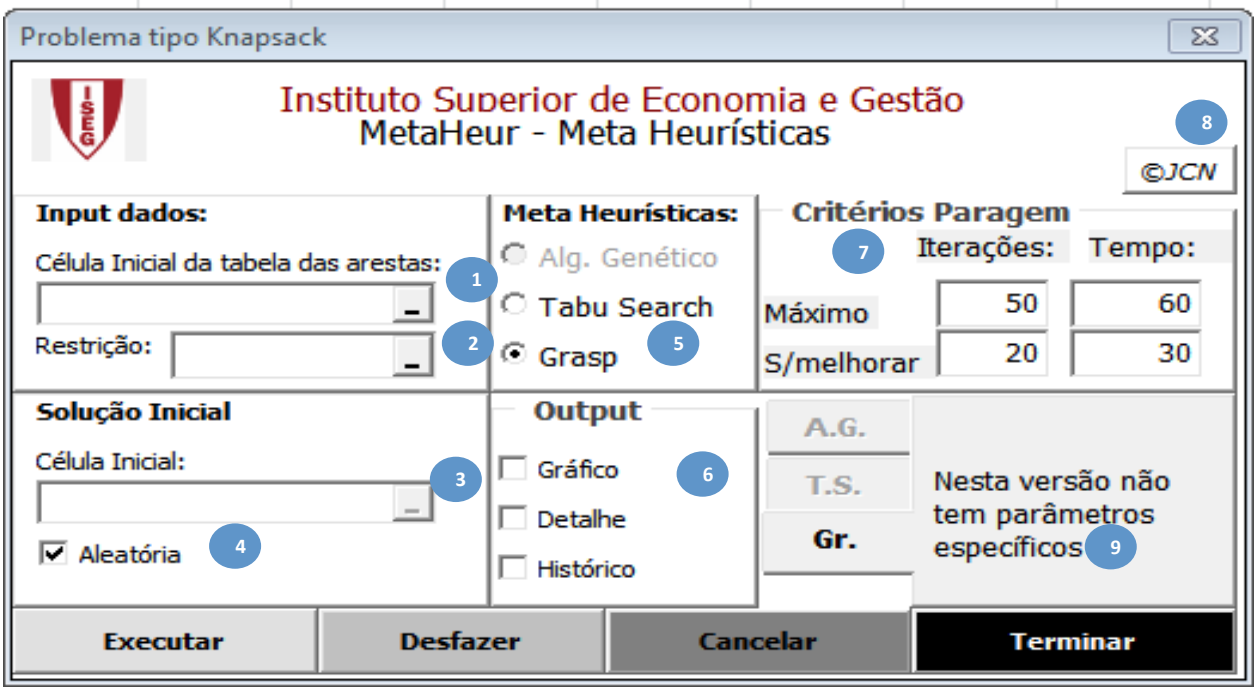


Figura 19 - Painel problema do Saco-*Mochila* metaheurística GRASP

Explicam-se de seguida os símbolos numerados da Figura 19

1 Componente que permite seleccionar a célula inicial da tabela dos arcos. Por exemplo, se a tabela dos arcos ocupar a range A1:C200, será a célula A1 que deverá ser seleccionada. O programa pressupõe que a tabela dos arcos não contém espaços em branco e fará o carregamento da tabela toda até encontrar o primeiro espaço em branco, que para o programa significa que não existe mais nenhum elemento para ser carregado. A referida tabela deverá conter na primeira coluna a identificação do elemento, na segunda coluna o grau de satisfação e na terceira coluna o custo ou o peso desse elemento.

2 Componente que se deve usar para seleccionar a célula que contém o termo independente da restrição, ou seja por exemplo, o custo ou o peso total que não pode ser ultrapassado.

3 Pode-se optar por introduzir uma solução inicial ou por se fazer a sua geração aleatória. Caso pretenda introduzir uma solução inicial dever-se-á clicar no componente intitulado Aleatório 4, de modo a não ter a *flag* seleccionada para o valor *verdadeiro*. Desta forma, o componente de leitura da solução passará a estar activo. Com este componente activo deverá ser seleccionada a célula inicial de uma coluna que está relacionada com a coluna de identificação dos elementos através do seu número de ordem que deverá conter apenas valores binários, isto é, para cada elemento o valor 1 indica que pertence à solução inicial e o valor 0 que não pertence. Serão efectuados alguns controlos para validar a solução proposta, como por exemplo, pedir ao utilizador para corrigir um valor encontrado que não seja binário.

4 Por defeito, a opção passa pela geração aleatória da solução inicial, caso contrário deverá remeter-se ao ponto anterior referente à introdução da solução inicial. Caso opte pela geração aleatória da solução inicial bastará assegurar que a *flag* do componente Aleatório esteja activa.

5 Painel que permite ao utilizador seleccionar a meta heurística adequada. Apenas se encontram disponíveis as metaheurísticas *GRASP* e *Tabu Search*. Não está ainda disponível a que recorre a um Algoritmo Genético e por essa razão esta opção não está activa. Conforme a selecção que se efectua relativa à metaheurística, os parâmetros específicos da metaheurística seleccionada são exibidos em 9. Concluindo, de acordo com a metaheurística seleccionada surge um quadro de parâmetros adequados.

6 Podem ser seleccionados alguns tipos de *outputs* adicionais. Ao seleccionar Gráfico, será efectuado um gráfico com a evolução das soluções obtidas. Se seleccionar Detalhe, serão exibidas todas as soluções obtidas incluindo as rejeitadas. Se seleccionar Histórico, será guardada a solução obtida para o problema numa folha criada para o efeito, onde é exibida a solução inicial e a solução final obtida. Esta função não estará disponível de imediato.

7 Neste quadro são exibidos os critérios de paragem da metaheurística e o primeiro destes critérios que se verificar, vai executar o procedimento de paragem da metaheurística. Estes critérios estão divididos em número de iterações (ou seja, o número de tentativas de busca de uma solução melhor que a inicial) ou por um tempo máximo decorrido desde o início da busca de uma solução melhor, em segundos. Alternativamente a sua execução poderá parar caso seja atingido um certo número de iterações sem melhorar a solução actual obtida (isto é, no caso de uma determinada solução ser obtida, não se consegue obter uma solução melhor ao fim de um certo número de tentativas), ou o seu tempo de busca se tenha esgotado sem ter havido nenhuma melhoria na solução actual obtida.

8 Botão que exibe informações sobre este trabalho, os orientadores deste trabalho e o seu autor.

9 De acordo com a metaheurística seleccionada no painel 5 serão exibidos parâmetros específicos dessa meta heurística. Por exemplo, caso seja seleccionada uma GRASP não haverá parâmetros adicionais, mas no caso de seleccionar a *Tabu Search* então terão de ser especificados os seguintes parâmetros: Coeficiente de Diversificação, taxa que permite fazer a busca de uma solução fora da vizinhança da solução actual, ou seja, a frequência com que que vai diversificar a busca fora da vizinhança da solução actual; Tamanho da Lista Tabu, ou seja o tamanho da sua memória, define o tamanho do número das soluções onde se efectuaram as últimas buscas.

6.7. Exemplo do problema do Saco-*Mochila* (knapsack):

Problema

Para incentivar a frequência de um curso de Verão de investigação operacional os organizadores dispuseram-se a pagar as despesas com a deslocação dos participantes, para o que têm um fundo de 6000 unidades monetárias (u.m.).

Os candidatos que se apresentam na Tabela 2 submeteram-se a provas de admissão. As classificações de cada candidato, bem como os respectivos custos de deslocação em unidades monetárias, também constam da tabela 2.

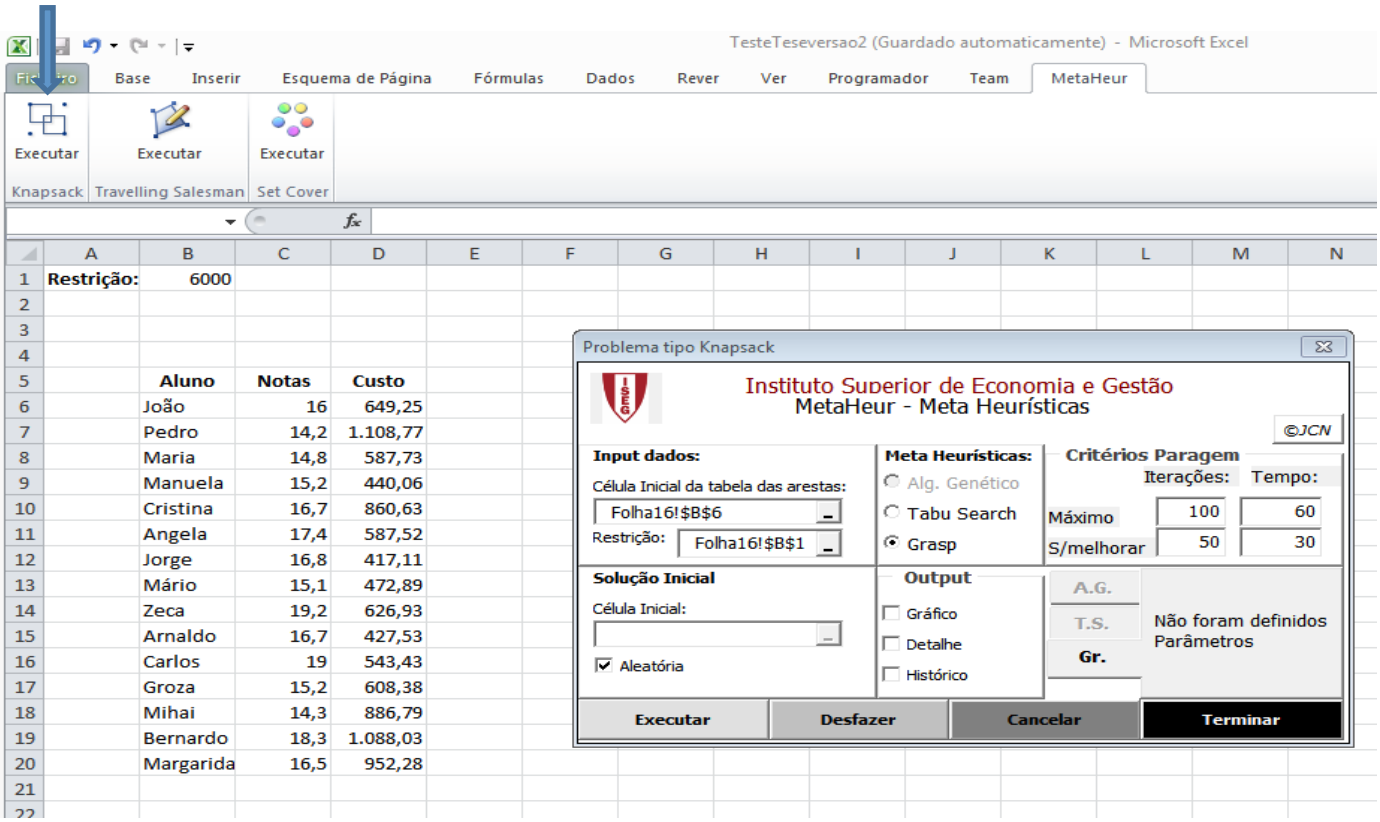
Nome Aluno	Classificações	Custos
João	16,6	649,25
Pedro	14,2	1108,77
Maria	14,8	587,73
Manuela	15,2	440,06
Cristina	16,7	860,63
Angela	17,4	587,52
Jorge	16,8	417,11
Mário	15,1	472,89
Zeca	19,2	626,93
Arnaldo	16,7	427,53
Carlos	19	543,43
Groza	15,2	608,38
Mihai	14,3	886,79
Bernardo	18,3	1008,03
Margarida	16,5	952,28

TABELA 2 -DADOS PARA O PROBLEMA KNAPSACK

Com base na informação disponível, os organizadores pretendem determinar os candidatos a seleccionar para o curso de Verão, sem ultrapassar o montante disponível para deslocações.

Resolução usando algoritmo *GRASP*:

Para o efeito e partindo do pressuposto de que o MetaHeur está já instalado como um suplemento do Excel, bastará inserir as tabelas dadas numa qualquer folha de um livro, de acordo com a seguinte figura:



	A	B	C	D
1	**Restrição:**	6000		
2				
3				
4				
5		**Aluno**	**Notas**	**Custo**
6		João	16	649,25
7		Pedro	14,2	1.108,77
8		Maria	14,8	587,73
9		Manuela	15,2	440,06
10		Cristina	16,7	860,63
11		Angela	17,4	587,52
12		Jorge	16,8	417,11
13		Mário	15,1	472,89
14		Zeca	19,2	626,93
15		Arnaldo	16,7	427,53
16		Carlos	19	543,43
17		Groza	15,2	608,38
18		Mihai	14,3	886,79
19		Bernardo	18,3	1.088,03
20		Margarida	16,5	952,28
21				

FIGURA 20 - RESOLUÇÃO PROB. SACO-MOCHILA COM GRASP

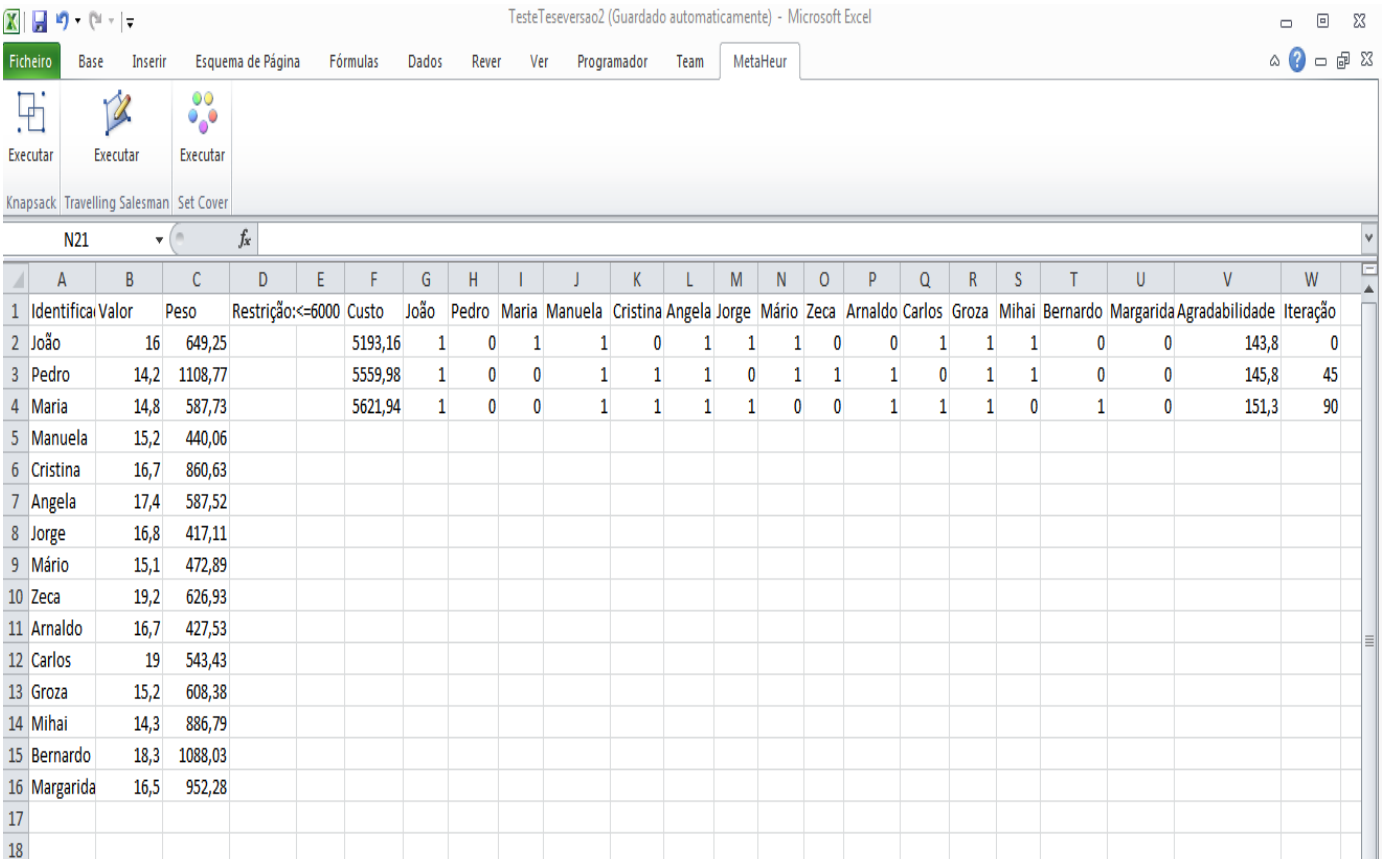
Deve seleccionar no friso, o separador MetaHeur e seleccionar, como indicado na Figura 20, o *icon* correspondente ao tipo de problema do Saco-Mochila (Knapsack). Depois de aberto o painel, deve indicar a célula inicial da tabela, a célula onde está a restrição, seleccionar o algoritmo, o Output que vamos deixar em branco e pedir que seja gerada uma solução inicial.

Após o painel estar preenchido, caso haja problemas poderemos pressionar o botão desfazer para voltar a inserir novamente as células, caso contrário, basta pressionar em Executar para a resolução do problema, nas condicionantes exibidas no painel do MetaHeur, Figura 20.

Para a exibição do Output, o Metaheur vai adicionar para esse efeito uma nova folha ao livro onde estão os dados inseridos e como não foi pedido o seu detalhe no output, as soluções rejeitas não serão apresentadas.

A primeira linha vai apresentar a solução inicial com o número de iteração zero.

As primeiras três colunas, mostram a forma como os dados foram lidos, a restrição que se entende como sendo a capacidade do Saco-Mochila.



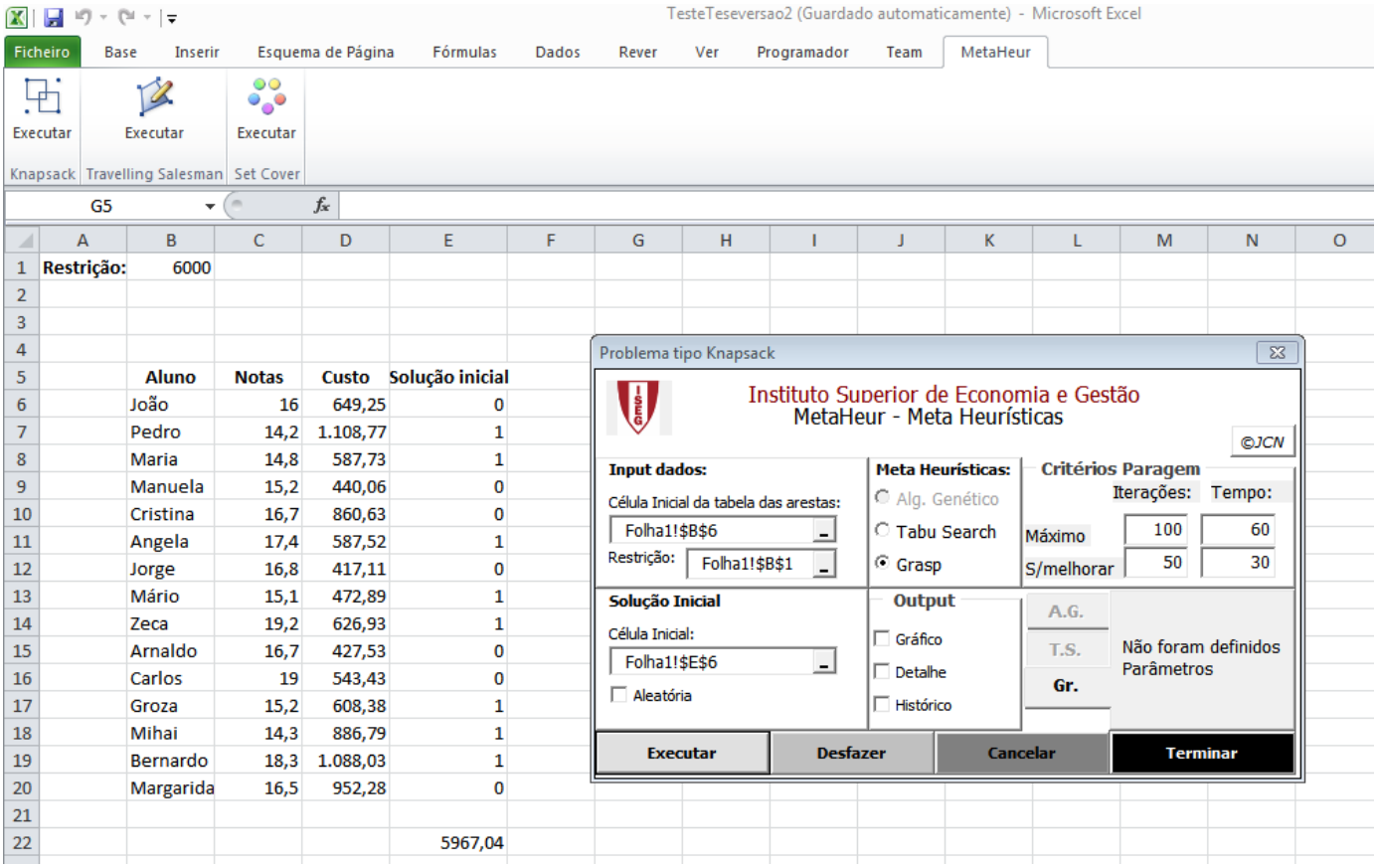
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	Identifica	Valor	Peso	Restrição:<=6000		Custo	João	Pedro	Maria	Manuela	Cristina	Angela	Jorge	Mário	Zeca	Arnaldo	Carlos	Groza	Mihai	Bernardo	Margarida	Agradabilidade	Iteração
2	João	16	649,25			5193,16	1	0	1	1	0	1	1	1	0	0	1	1	1	0	0	143,8	0
3	Pedro	14,2	1108,77			5559,98	1	0	0	1	1	1	0	1	1	1	0	1	1	0	0	145,8	45
4	Maria	14,8	587,73			5621,94	1	0	0	1	1	1	1	0	0	1	1	1	0	1	0	151,3	90
5	Manuela	15,2	440,06																				
6	Cristina	16,7	860,63																				
7	Angela	17,4	587,52																				
8	Jorge	16,8	417,11																				
9	Mário	15,1	472,89																				
10	Zeca	19,2	626,93																				
11	Arnaldo	16,7	427,53																				
12	Carlos	19	543,43																				
13	Groza	15,2	608,38																				
14	Mihai	14,3	886,79																				
15	Bernardo	18,3	1088,03																				
16	Margarida	16,5	952,28																				
17																							
18																							

FIGURA 21 - OUTPUT GRASP PARA KNAPSACK

Como não foi escolhida a opção detalhe para o output, apenas teremos as soluções aceites e na última linha a solução binária (1 para seleccionado, 0 para o contrário) a que o algoritmo chegou.

Assim, da leitura do Output gerado, a solução inicial gerada aleatoriamente tinha um custo de 5.193,16, para uma agradabilidade de 143,8 e foi melhorada para uma agradabilidade de 151,3, correspondente a um custo superior de 5.621,94 ao fim de 90 iterações. Passando a solução por incluir o João, Manuela, Cristina, Angela, Jorge, Arnaldo, Carlos, Groza e Bernardo.

Este problema também poderia ter sido resolvido, sem fazer gerar uma solução inicial. Caso pretendesse incluir logo uma solução pré-determinada, poderia colocar as tabelas e a solução inicial numa qualquer folha do livro conforme a figura seguinte:



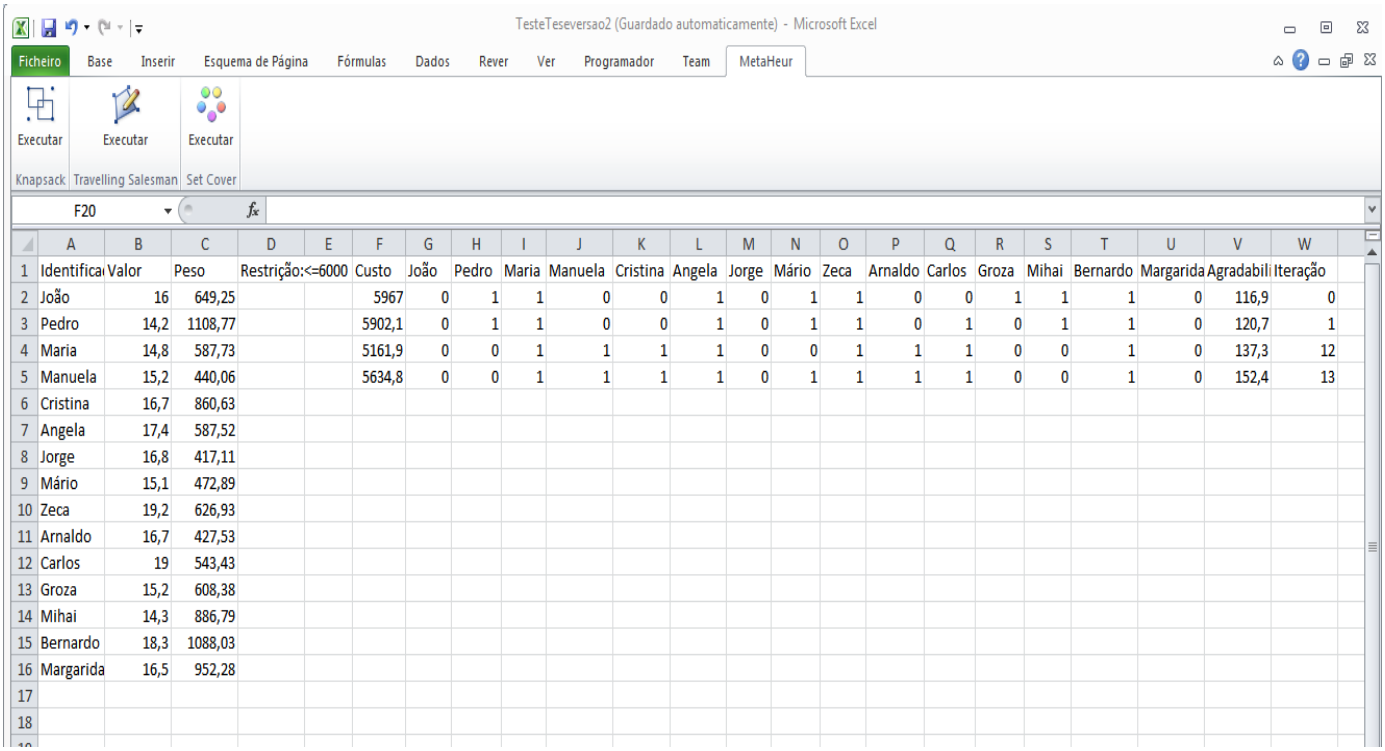
	A	B	C	D	E
1	**Restrição:**	6000			
2					
3					
4					
5		**Aluno**	**Notas**	**Custo**	**Solução inicial**
6		João	16	649,25	0
7		Pedro	14,2	1.108,77	1
8		Maria	14,8	587,73	1
9		Manuela	15,2	440,06	0
10		Cristina	16,7	860,63	0
11		Angela	17,4	587,52	1
12		Jorge	16,8	417,11	0
13		Mário	15,1	472,89	1
14		Zeca	19,2	626,93	1
15		Arnaldo	16,7	427,53	0
16		Carlos	19	543,43	0
17		Groza	15,2	608,38	1
18		Mihai	14,3	886,79	1
19		Bernardo	18,3	1.088,03	1
20		Margarida	16,5	952,28	0
21					
22					5967,04

FIGURA 22 - GRASP PARA KNAPSACK C/ SOLUÇÃO INICIAL

A solução inicial deve estar na forma de um vector binário, relacionada com a matriz de dados. Por exemplo, o Pedro, com 14,2 e um custo de 1.108,77 faz parte da solução inicial.

O utilizador tem a liberdade de colocar o vector (em coluna sempre) em qualquer parte da folha, não precisa de estar contíguo à tabela de dados, isto é, poderia estar da célula F3 à F17. O único cuidado a ter é deixar uma célula em branco após a matriz de dados e também no vector que representa a solução inicial. Por exemplo, se o custo que está na célula E22, estivesse em B21 ou em E21, impediria a leitura correcta dos dados. É fundamental que tenha se sempre esse cuidado.

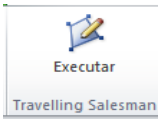
Neste caso e como os dados representados como estão na Figura 22, com a indicação da célula inicial da solução final e eliminando a *flag* do aleatório, o MetaHeur pode ser executado, obtendo-se o seguinte output, com a mesma estrutura da Figura 21, apenas com a diferença de os dados:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	Identifica	Valor	Peso	Restrição:<=6000		Custo	João	Pedro	Maria	Manuela	Cristina	Angela	Jorge	Mário	Zeca	Arnaldo	Carlos	Groza	Mihai	Bernardo	Margarida	Agradabili	Iteração
2	João	16	649,25			5967	0	1	1	0	0	1	0	1	1	0	0	1	1	1	0	116,9	0
3	Pedro	14,2	1108,77			5902,1	0	1	1	0	0	1	0	1	1	0	1	0	1	1	0	120,7	1
4	Maria	14,8	587,73			5161,9	0	0	1	1	1	1	0	0	1	1	1	0	0	1	0	137,3	12
5	Manuela	15,2	440,06			5634,8	0	0	1	1	1	1	0	1	1	1	1	0	0	1	0	152,4	13
6	Cristina	16,7	860,63																				
7	Angela	17,4	587,52																				
8	Jorge	16,8	417,11																				
9	Mário	15,1	472,89																				
10	Zeca	19,2	626,93																				
11	Arnaldo	16,7	427,53																				
12	Carlos	19	543,43																				
13	Groza	15,2	608,38																				
14	Mihai	14,3	886,79																				
15	Bernardo	18,3	1088,03																				
16	Margarida	16,5	952,28																				
17																							
18																							

FIGURA 23 - OUTPUT GRASP PARA KNAPSACK C/ SOLUÇÃO INICIAL

6.8. Problema do Caixeiro-Viajante (TSP)

7. Caso se pretenda resolver um problema do tipo Caixeiro-Viajante, deve pressionar-se o botão correspondente que se encontra no separador *Travelling Salesman* do friso do MetaHeur:
8. 

9. É exibido o seguinte painel:

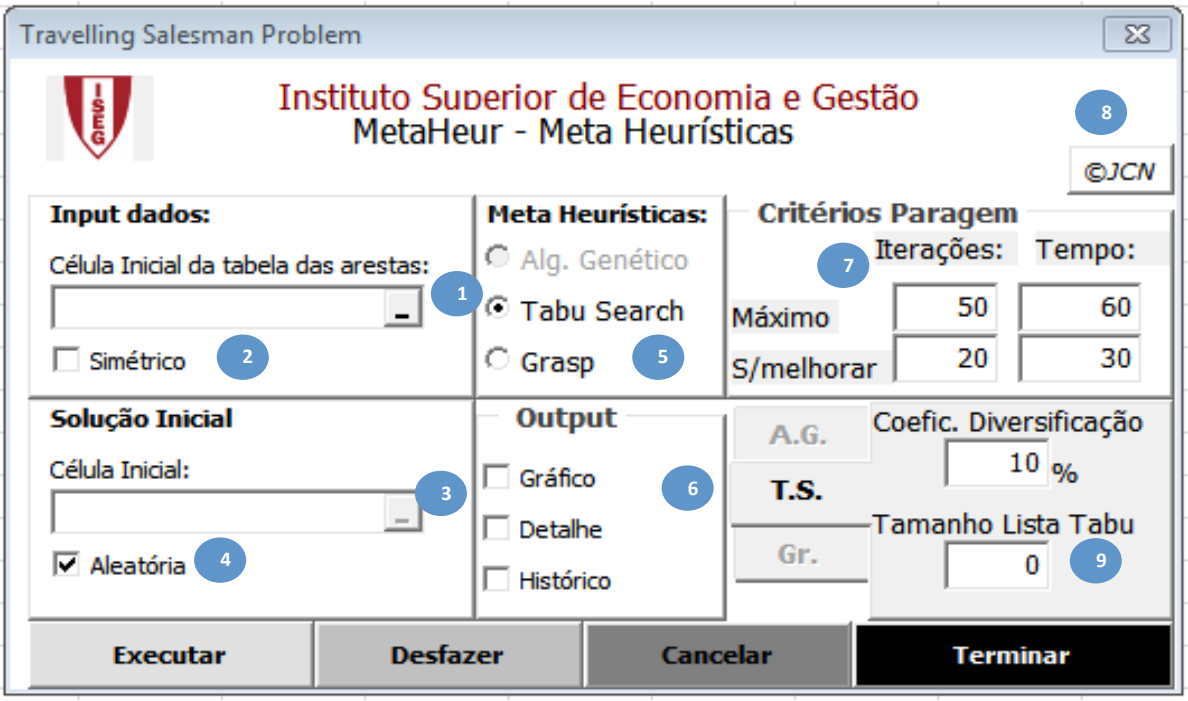


FIGURA 24 - PAINEL METAHEUR PARA O TSP METAHEURÍSTICA TABU-SEARCH

Explicam-se de seguida os símbolos numerados na Figura 24:

1 Componente que permite seleccionar a célula inicial da tabela dos arcos. Por exemplo, se a tabela dos arcos ocupar a range A1:C200, será a célula A1 que deverá ser seleccionada. O programa pressupõe que a tabela dos arcos não contém espaços em branco e fará o carregamento da tabela toda até encontrar o primeiro espaço em branco, que para o programa significa que não existe mais nenhum elemento para ser carregado. A referida tabela deverá conter na primeira coluna o vértice de origem do arco, na segunda coluna o vértice de destino do arco e na terceira coluna a distância ou o peso desse arco.

2 Componente que se deve usar no caso do problema do Caixeiro-Viajante ser simétrico. Se esta opção for activada o MetaHeur vau duplicar todos os arcos definidos no ponto anterior, isto é, para cada arco lido, cria um novo onde troca os vértices, o de origem passa a destino e vice-versa.

3 Pode-se optar por introduzir uma solução inicial ou por se fazer a sua geração aleatória. Caso pretenda introduzir uma solução inicial dever-se-á clicar no componente intitulado Aleatório 4, de modo a não ter a *flag* seleccionada para o valor *verdadeiro*. Desta forma, o componente de leitura da solução passará a estar activo. Com este componente activo deverá se seleccionada a célula inicial de uma coluna que está relacionada com a coluna de identificação dos elementos através do seu número de ordem que deverá conter apenas valores binários, isto é, para cada elemento o valor 1 indica que pertence à solução inicial e o valor 0 que não pertence. Serão efectuados alguns controlos para validar a solução proposta, como por exemplo, pedir ao utilizador para corrigir um valor encontrado que não seja binário.

4 Por defeito, a opção passa pela geração aleatória da solução inicial, caso contrário deverá remeter-se ao ponto anterior referente à introdução da solução inicial. Caso opte pela geração aleatória da solução inicial bastará assegurar que a *flag* do componente Aleatório esteja activa.

5 Painel que permite ao utilizador seleccionar a meta heurística adequada. Apenas se encontram disponíveis as meta heurísticas *GRASP* e *Tabu Search*. Não está ainda disponível a que recorre a um Algoritmo Genético e por essa razão esta opção não está activa. Conforme a selecção que se efectua relativa à metaheurística, os parâmetros específicos da metaheurística seleccionada são exibidos em 9. Concluindo, de acordo com a metaheurística seleccionada surge um quadro de parâmetros adequados.

6 Podem ser seleccionados alguns tipos de *outputs* adicionais. Ao seleccionar Gráfico, será efectuado um gráfico com a evolução das soluções obtidas. Se seleccionar Detalhe, serão exibidas todas as soluções obtidas incluindo as rejeitadas. Se seleccionar Histórico, será guardada a solução obtida para o problema numa folha criada para o efeito, onde é exibida a solução inicial e a solução final obtida. Esta função não estará disponível de imediato.

7 Neste quadro são exibidos os critérios de paragem da metaheurística e o primeiro destes critérios que se verificar, vai executar o procedimento de paragem da metaheurística. Estes critérios estão divididos em número de iterações (ou seja, o número de tentativas de busca de uma solução melhor que a inicial) ou por um tempo máximo decorrido desde o início da busca de uma solução melhor, em segundos. Alternativamente a sua execução poderá parar caso seja atingido um certo número de iterações sem melhorar a solução actual obtida (isto é, no caso de uma determinada solução ser obtida, não se consegue obter uma solução melhor ao fim de

um certo número de tentativas), ou o seu tempo de busca se tenha esgotado sem ter havido nenhuma melhoria na solução actual obtida.

8 Botão que exibe informações sobre este trabalho, os orientadores deste trabalho e o seu autor.

9 De acordo com a metaheurística seleccionada no painel 5 serão exibidos parâmetros específicos dessa meta heurística. Por exemplo, caso seja seleccionada uma GRASP não haverá parâmetros adicionais, mas no caso de seleccionar a *Tabu Search* então terão de ser especificados os seguintes parâmetros: Coeficiente de Diversificação, taxa que permite fazer a busca de uma solução fora da vizinhança da solução actual, ou seja, a frequência com que que vai diversificar a busca fora da vizinhança da solução actual; Tamanho da Lista Tabu, ou seja o tamanho da sua memória, define o tamanho do número das soluções onde se efectuaram as últimas buscas.

9.1. Exemplo do problema do Caixeiro-Viajante (TSP):

Problema:

Imagine que tem de organizar uma deslocação a um conjunto de cidades Ibéricas, e estabeleceu as distâncias entre as referias cidades:

Badajoz	**Évora**	**99**
Badajoz	Córdova	269
Faro	Badajoz	331
Aveiro	Badajoz	372
Madrid	Córdova	424
Évora	Madrid	502
Aveiro	Faro	582
Faro	Madrid	750
Évora	Faro	231
Faro	Córdova	326

Aveiro	Évora	353
Badajoz	Madrid	403
Évora	Córdova	426
Aveiro	Madrid	559
Aveiro	Córdova	641

Indique qual o circuito ideal para percorrer estas cidades, sem repetição, minimizando a distância a percorrer?

Resolução usando algoritmo *Tabu-Search*

A estrutura de apresentação dos dados é igual à do problema anterior e parece ser a forma mais intuitiva de o fazer.

Deve seleccionar no friso, o separador MetaHeur e pressionar o *icon* correspondente ao tipo de problema Travelling Salesman (Caixeiro-Viajante). Depois de aberto o painel, deve indicar a célula inicial da tabela, se a tabela é simétrica (por exemplo, se de Aveiro para Faro são 582 km, o sistema deve considerar que de Faro para Aveiro também se pode ir e a distância é também de 582 km, neste caso se a opção estiver activada, o MetaHeur constrói os arcos em falta, poupando ao utilizador esse trabalho), seleccionar o algoritmo, o Output que vamos deixar em branco e pedir que seja gerada uma solução inicial.

No caso de a solução não ser simétrica, deveria ser retirada a *flag* da opção simétrica e o MetaHeur limitava-se a ler os arcos que tivessem sido introduzidos.

No caso dos parâmetros do Tabu-Search escolhe-se um coeficiente relativamente alto para a dispersão, 20%, e uma memória de curto prazo (lista Tabu) de três movimentos.

O MetaHeur de certa forma, valida dados introduzidos que por lapso possam estar errados, como por exemplo um coeficiente de dispersão superior a 100%, ou uma lista tabu superior em tamanho ao número de vértices, evitando assim que o utilizador por este facto, tenha de introduzir os dados novamente.

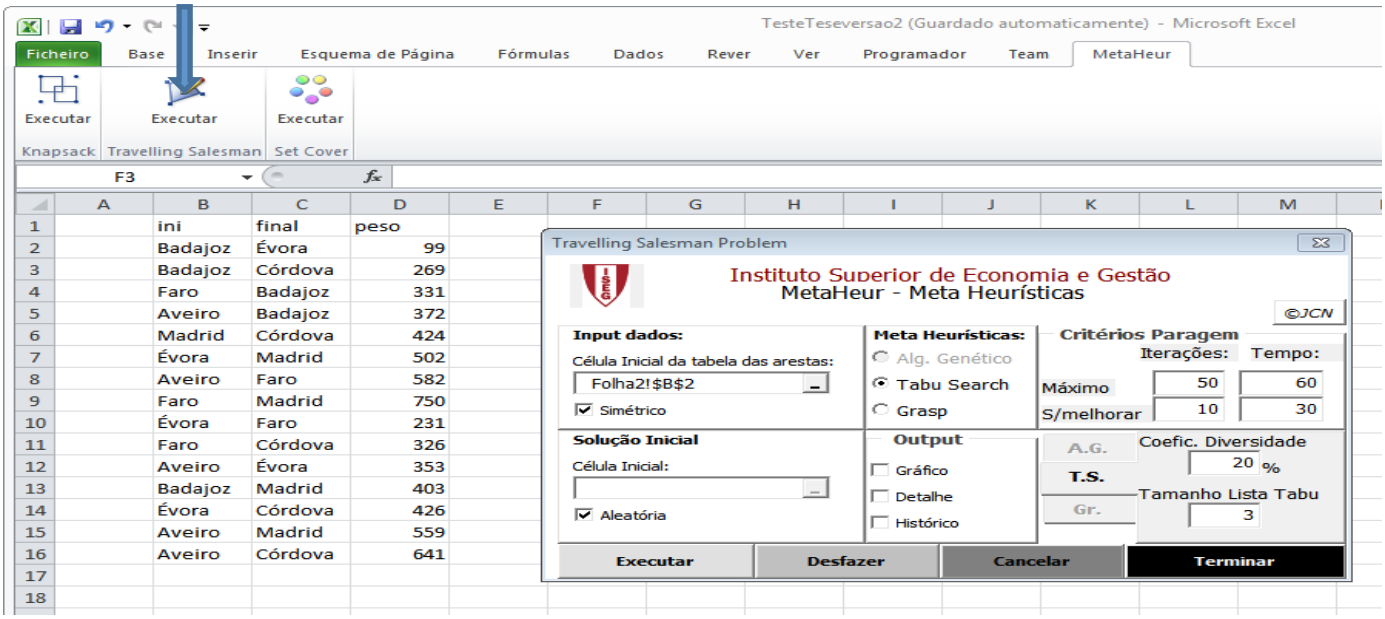


FIGURA 25 - RESOLUÇÃO TSP C/ TABU-SEARCH

Como Output, à semelhança do problema anterior, é criada uma nova folha que apresenta a distância e o circuito que tem de ser percorrido:

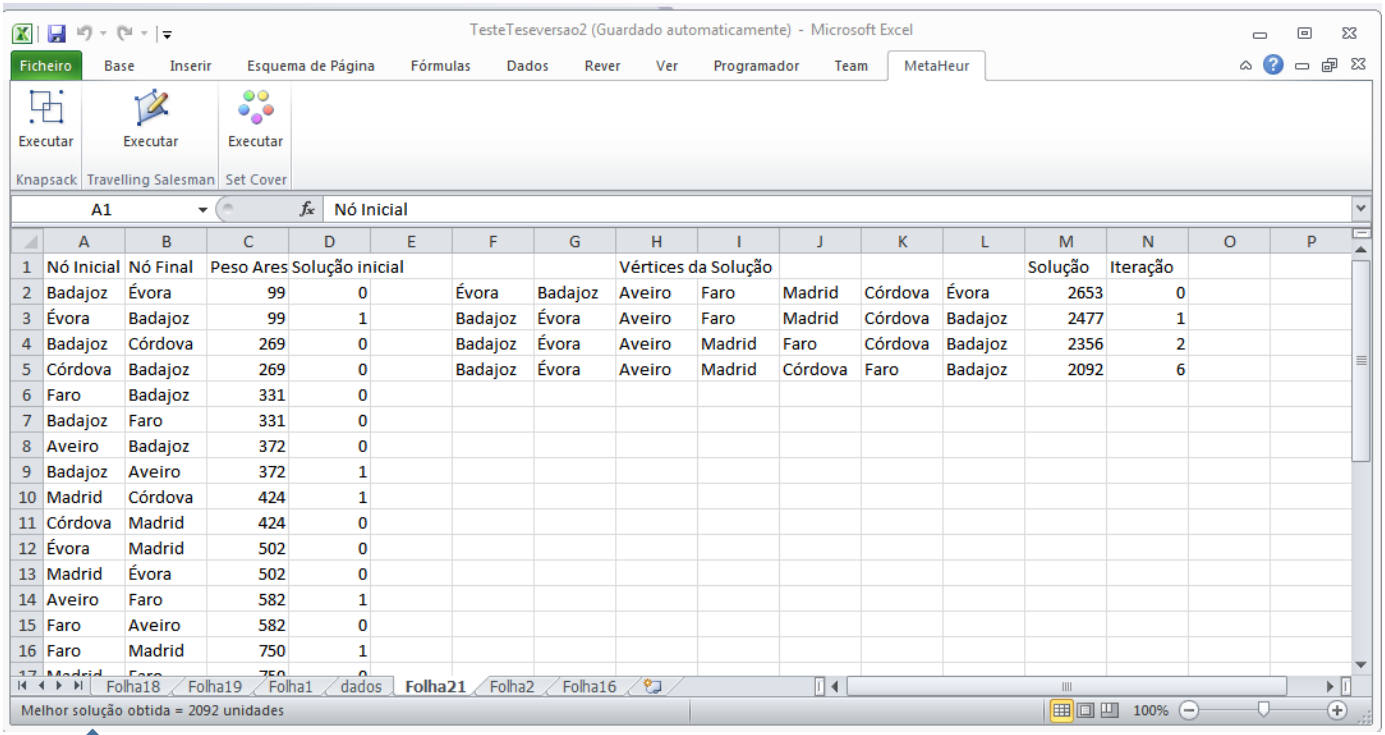


FIGURA 26 - OUTPUT DA RESOLUÇÃO TSP C/ TABU-SEARCH

A barra de estado do Excel, é utilizada para dar informações também.

10. CONCLUSÕES

A maior parte dos objectivos propostos foram atingidos.

Considerando os prazos e o âmbito deste trabalho não houve tempo para se optimizar ainda mais o código e/ou procurar soluções alternativas. Talvez por essa razão é que normalmente são empresas a desenvolver este tipo de programas.

Penso que este trabalho poderia ter sido desenvolvido por mais do que um mestrando, funcionando em equipa e coordenados por um Professor.

Mas se pensarmos que este é um trabalho que poderá ser visto como o de arranque para um projecto que envolva mais pessoas, ou para um prazo mais alargado, então é possível pensar como sendo um trabalho bastante promissor e interessante. Por esta razão é que o código está estruturado em módulos independentes, excepto uma ou outra rotina que usa funções que estão descritas noutro módulo, em geral manteve-se uma certa estanquicidade entre os módulos.

O módulo para os problemas de cobertura só tem uma caixa de mensagem e é muito fácil o seu desenvolvimento por terceiros.

A codificação das rotinas relativas ao algoritmo genético já terá de ser feito considerando os desenvolvimentos até aqui efectuados. Em nossa opinião este trabalho é fundamental para esse efeito.

O signatário estará sempre disponível para dar apoio aos futuros colegas que pretendam desenvolver o MetaHeur.

BIBLIOGRAFIA

Epp, S.L (2004), *Discrete Mathematics with Applications*, 3nd edn, Brooks/Cole,Belmonte, USA.

Feo, T.A. e Resende, M.C.G. (1995), Greedy Randomize Adaptive Search Procedures. *Journal of Global Optimization*, 6:109-133.

Glover, F. (1989), Tabu Search – Part I. *ORSA Journal on Computing* (190-206).

Goldberg, D. (1989), *Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Professional,* USA.

Green, J., Bullen, S., Bovey, R. e Alexander, M. (2007), *Excel 2007 VBA Programmer's Reference*, Wiley, Indianapolis, USA.

Hillier, F.S. e Lieberman, G.J. (2010), *Introduction to Operations Research* , 9nd edn, McGraw-Hill, International Editiom, New York.

Russel, S. e Norvig, P. (2004), *Inteligência Artificial*, 4 nd edn, Elsevier, Rio de Janeiro, Brasil.

http://www.sorting-algorithms.com/quick-sort-3-way consultado em 31/01/2011

O Mestrando

Nº 11620 Jorge Oliveira da Costa Neves